GALS Design Methodology Based on Pausible Clocking

DISSERTATION

zur Erlangung des akademischen Grades

Dr.–Ing.
im Fach Informatik

eingereicht an der
Mathematisch-Wissenschaftlichen Fakultät II
Humboldt-Universität zu Berlin

von
M. Eng. Xin Fan

Präsident der Humboldt-Universitat zu Berlin:
Prof. Dr. Jan-Hendrik Olbertz

Dekan der Mathematisch-Wissenschaftlichen Fakultät II:
Prof. Dr. Elmar Kulke

Gutachter:

Prof. Dr. Eckhard Grass, Humboldt-Universität zu Berlin
Prof. Dr. Andreas Steininger, Technische Universität Wien
Prof. Dr. Heinrich Vierhaus, Brandenburgische Technische Universität

eingereicht am: 06 September, 2013
Tag der mündlichen Prüfung: 19 Dezember, 2013

Acknowledgements

I would like to express my sincere gratitude to my supervisor Prof. Eckhard Grass and also to my project leader Dr. Milos Krstic. Discussions with them shaped the work presented in this thesis. It was the practical supports of Milos to make all the test chips possible. I really appreciate the thorough proofreading made by Eckhard on the draft of my thesis. Without their complete trust, I would never have accomplished this work.

To some extent, IEEE *Xplore* is my teacher. I enjoy the working environment in IHP. Actually, I am impressed by the excellent organization of Germany in research.

Special thanks to the test group in my department. Their professional work on chip test and measurements contributed a lot to the soundness of my work.

I am grateful to my friends and "lunch mates" in IHP, Oliver Schrape, Frank Vater, and Steffen Zeidler, especially for their kindness to my ignorance on Germen.

Life would be boring without my Chinese colleagues: Dr. Yaoming Sun and Ruoyu Wang. Thanks to them for many pleasant moments we spent together.

This thesis is dedicated to my dear parents, Mr. Zhonggang Fan and Mrs. Suhuai Li, and my lovely wife, Mrs. Leli Zhang. Also, it is in memory of my passed youth.

Abstract

Globally asynchronous locally synchronous (GALS) design presents a solution of scalability and modularity to SoC integration. Today, it has been widely applied in the industry. Most of the GALS systems are based on dual-clock FIFOs for clock domain crossing. To avoid performance loss due to synchronization latency, the on-chip FIFOs need to be sufficiently large. This, however, often leads to considerable hardware costs. Efficient design solutions of GALS are therefore in great demand.

Data synchronization by pausible clocking presents an alternative of GALS design to the dual-clock FIFO based solutions. For safe synchronization it adapts, if necessary, the clock transitions according to the arrival time of input data. Clock domain crossing is thus elegantly addressed. It intrinsically allows the GALS design with both, low synchronization latency and little hardware overhead. Plenty of research has been carried out, especially from the academy, for this reason.

This thesis reports our latest progress in GALS design bases on pausible clocking. Critical design issues on synchronization reliability and communication performance are studied systematically and analytically. A loosely-coupled GALS data-link design is proposed. It supports metastability-free synchronization for sub-cycle clock-tree delay, and accommodates continuous data transfer for high-throughput communication. Only marginal costs of power and silicon area are required.

GALS design has been employed to cope with the on-chip digital switching noise in our work. Plesiochronous clocking with power-consumption balance between GALS blocks is in particular explored. Given M clock domains, a reduction of $20\lg M$ dB on the spectral peaks of supply current at the fundamental clock frequency is theoretically derived. In comparison with the existing synchronous design solutions, it thus presents an alternative to effective attenuation of digital switching noise.

The developed GALS design methodology has been applied to chip implementation. Two complicated industry-relevant test chips, named Lighthouse and Moonrake, were designed and fabricated using state-of-the-art technologies. The experimental results as well as the on-chip measurements are reported here in detail. We expect that, our work will contribute to the practical applications of GALS design based on pausible clocking in the industry.

Keywords: GALS, asynchronous circuits, pausible clocking, digital switching noise

Zusammenfassung

Globally Asynchronous Locally Synchronous (GALS) Design ist eine Lösung zur Skalierbarkeit und Modularität für die SoC-Integration. Heutzutage ist GALS-Design weit in der Industrie angewendet. Die meisten GALS-Systeme basieren auf Dual-Clock-FIFOs für die Kommunikation Zwischen Taktdomänen. Um Leistungsverluste aufgrund der Synchronisationslatenzzeit zu vermindern, müssen die On-Chip-FIFOs ausreichend groß sein. Dies führt jedoch oft zu erheblichen Kosten-Hardware. Effiziente GALS-Lösungen sind daher vonnöten.

Daten synchronisation mittels der Pausierenden Taktung präsentiert ein alternatives GALS Design zu den Dual-Clock-FIFO-basierten Lösungen. Es passt gegebenenfalls die Takt-übergänge in Übereinstimmung mit der Ankunftszeit von Eingangsdaten für eine sichere Synchronisationan. Clock Domain Crossing ist damit elegant adressiert. Intrinsisch ermöglicht es das GALS Design sowohl mit geringer Synchronisation-slatenz als auch wenigen Hardware-Kosten.

Diese Arbeit berichtet unsere neuesten Fortschritte in GALS Design, das auf der Pausierenden Taktung basiert. Kritische Designthemen in Bezug auf Synchronisation-szuverlässigkeit bzw. Kommunikationsfähigkeit sind systematisch und analytisch untersucht. Ein lose gekoppeltes GALS Data-Link-Design wird vorgeschlagen. Es unterstützt metastabilitätsfreie Synchronisation für Sub-Takt-Baum Verzögerungen. Außerdem unterstützt es kontinuierliche Datenübertragung für High-Throughput-Kommunikation. Die Rosten hinsichtlich Energie verbrauch und Chipfläche sind marginal.

GALS Design ist eingesetzt, um digitales On-Chip Umschaltrauschen zu verringern. Plesiochron Taktung mit balanciertem Leistungsverbrauch zwischen GALS Blöcken wird insbesondere untersucht. Für M Taktbereiche wird eine Reduzierung um 20lgM dB für die spektralen Spitzen des Versorgungsstroms bei der Takt-Grundfrequenz theoretisch hergcleitet. Im Vergleich zu den bestehenden synchronen Lösungen, geben diese Methode eine Alternative, um das digitale schaltrauschen effektiv zu senken.

Schließlich wurde die entwickelte GALS Design Methodik schon bei reale Chip-Implementierungen angewendet. Zwei komplizierte industriell relevante Test-Chips, *Lighthouse* und *Moonrake*, wurden entworfen und mit State-Of-The-Art-Technologien hergestellt. Die experimentellen Ergebnisse bzw. Chip-Messungen sind in dieser Arbeit im Detail berichtet. Wir hoffen, dass diese Arbeit zur industriellen Anwendungen des GALS Designs mit der Pausierenden Taktung beitragen wird.

Schlagwörter: GALS, asynchronen schaltungen, pausierend taktung, digitale umschalt-rauschen

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 GALS Design

Synchronous design has been prevailed in the VLSI industry over many decades. It is based on timing the switching activity of on-chip registers by a global clock signal. Delay constraints are imposed on combinational gates for the functional correctness. As a result, logic design and test can be greatly simplified. Mature CAD tools are available for modeling and implementation of synchronous circuits.

The downscaling of CMOS technology into the deep sub-micron regime, however, fundamentally challenges the synchronous design. Due to the increasing clock frequency and die size, it becomes hardware consuming to distribute the clock trees at negligible skew across a chip [1]. Process parameter variations and power integrity requirements further stretch the timing margins for timing analysis, giving rise to over conservative design [2]. Global synchronization also introduces difficulties on the integration of pre-developed IP-cores each working at an individual clock frequency [3].

On the opposite side are the advances in the asynchronous design. It was pioneered by David Huffman and David Muller [4] [5], and has attracted wide interest of research since Ivan Sutherland's work on micropipelines [6]. No clock signal is used for timing. Data processing is scheduled by the handshaking between neighbour logic. It naturally allows the design for scalability, reliability and modularity [7].

Although some feasible design flows have been proposed, the asynchronous design is still rarely applied in the industry [8]. Two primary issues account for the reason. The behavioral modeling and logic synthesis of asynchronous circuits fundamentally differ from the synchronous circuits. It is commonly believed that commercial-quality CAD tools for asynchronous design are not yet ready. Testability addresses another challenge. Asynchronous design complicates the chip test based on the standard ATEs (Automatic Test Equipment), which is, however, crucial for mass production.

Globally asynchronous locally synchronous (GALS) design aims at filling the gap between the synchronous and asynchronous paradigms. The concept of GALS was first introduced by Chapiro in 1984 [9]. Generally speaking, it refers to the design of SoCs with multiple local clocks. Figure 1.1 gives a conceptual abstract of GALS design. The entire system is partitioned into a number of GALS blocks. The functional modules of each block are timed by a local clock. Interconnects between blocks are realized by a-synchronous handshake circuits. This leads to an efficient decoupling of computation and communication in a GALS system. Depending on the design styles of GALS, the local clocks can be generated either internally or externally.

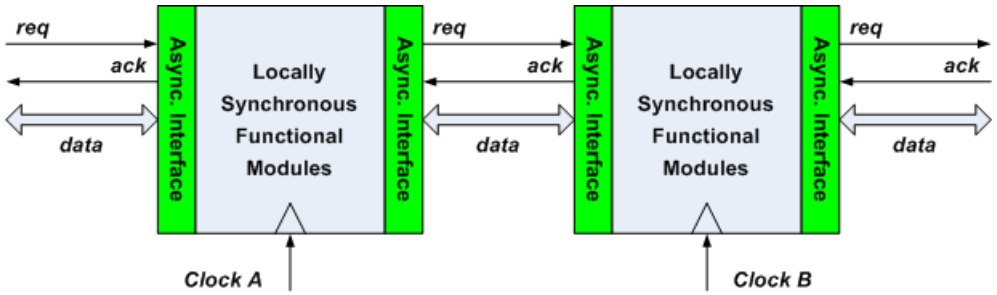


Figure 1.1 Conceptual abstract of GALS design

GALS design presents an alternative solution to system integration taking advantages of both the synchronous and the asynchronous designs. The majority of a GALS system is dominated by traditional synchronous circuits, and the asynchronous design is confined to the small portion of interface logic. Therefore, it allows the migration and reuse of mature design and test techniques of synchronous circuits. On the other side, it also facilitates to exploit the benefits of miniaturized CMOS process for large-scale and high-speed SoC designs. Today, GALS design is no longer a concept only: it has been widely applied in the modern VLSI industry.

1.2 Pausible Clocking

Asynchronous interface design poses the primary challenge for GALS integration. Up to now, most of the GALS systems are based on dual-clock FIFOs for clock domain crossing. An important reason lies in its full compatibility with the synchronous design flows. In this thesis, however, we will explore GALS design by pausible clocking. The pausible clocking scheme actually represents a “classic” design style of GALS. Its first prototype can be dated back to the work of Stucki and Cox in 1979 [10]. Rather than synchronizing tokens to the clocks, it adapts, if necessary, the transitions of local clocks for reliable synchronization.

Comparing with the popular FIFO-based solutions, the pausible clocking deserves our particular attention of research for two reasons. First, it is the only known technique, which can promise metastability-free synchronization between fully independent clocks. Second, it is considered to potentially afford high data-throughput with very little costs of power and silicon area. Both features make it attractive for advancing the design of GALS SoCs with performance and hardware efficiency.

This thesis reports the latest progress of our work on pausible clocking based GALS design. Design aspects critical for GALS performance are addressed systematically and analytically, including safe synchronization, high throughput communication and system partitioning. The experiments on two industry-relevant complex test chips, implemented using the developed GALS design methodology, are presented in great detail. We hope that, our work will contribute to the practical applications of GALS design by pausible clocking in the industry.

Chapter 2

Overview of GALS Interface Design

The interface circuit design imposes a main challenge on GALS system integration. This chapter briefs the design fundamentals of GALS interface circuits. A taxonomy of GALS design according to the timing relationships between local clocks is introduced. Primary design considerations on interface circuits, i.e., the synchronization reliability and overhead, are outlined. Representative GALS interface circuits, including boundary synchronizers, dual-clock FIFOs and pausible clocking, are overviewed. A comparison between typical GALS interface designs in terms of the major synchronization metrics is further addressed.

2.1 Taxonomy of GALS Design

Depending on the frequency relationship of the transmitter (TX) and receiver (RX) clocks, GALS designs can be classified into the following four categories [11] [12].

- Mesochronous clock design:
  In this case, TX and RX are working at exactly the same clock frequency, but have a stable (or slightly drifting) relative phase. Mesochronous design is often exploited by the hierarchical synthesis of clock distribution networks, where a number of tightly skewed local clock networks are triggered by a global source clock through an unbalanced clock distribution at the top level [13].
- Plesiochronous clock design:
  In this case, TX and RX are working at almost the same clock frequency, only with a tiny frequency mismatch. In the time domain, the deviation of frequency leads to an accumulated phase variation between local clocks. Gigabit Ethernet design is an example of plesiochronous clocking.
- Ratiochronous clock design:
  In this case, TX and RX are working at different clock frequencies, but obey the rational frequency relationship (such as 1:3 or 5:2). It results in a periodic and predicable phase difference between TX and RX clocks. For instance, frequency scaling (division or multiplication) by integer factors on a reference clock often presents a typical application of ratiochronous design.
- Asynchronous clock design:
  In this case, TX and RX are triggered by the local clocks which are uncorrelated with each other. No relative timing is assumed on clock frequency or phase.

Table II.1 Taxonomy of GALS design

	Synchronous	**Mesochronous**	**Plesiochronous**	**Ratiochronous**	**Asynchronous**
f_{TX}/f_{RX}	1	1	≈ 1	M/N	Any
$\emptyset_{TX} - \emptyset_{RX}$	Negligible	Constant	Drifting	Periodic	Variable

Table II.1 characterizes the clock frequency and phase relationships in four GALS design categories. Synchronous design style is also listed for comparison. In general the interface design for fully asynchronous clocking is suboptimal when applied to the other GALS communication scenarios, because no particular timing relationship is exploited. However, it addresses the universal solution and offers the maximum flexibility of local clock desynchronization. This indicates its fundamental importance for the applications of GALS design. Below we are going to highlight the practical difficulties and existing schemes of interconnection between fully asynchronous clock domains.

2.2 Synchronization Reliability and Overhead

2.2.1 Synchronization Reliability

It is well known that, to be safely sampled, an input signal has to be stable within a timing window (such as setup time and hold time) around the active clock edge of a latch. Once the input switches simultaneously with the clock edge, metastability occurs. Consequently, the latch output voltage can be indeterminate, neither high nor low [14]. In terms of GALS design, metastability is induced by the clock domain crossing (CDC). Since input signal are asynchronous with regard to the receiver clock, they can switch at any moment. This in fact imposes a great challenge for transferring data across clock domains in a GALS system.

Assuming that the arrival time of an input signal is uniformly distributed over the receiver clock period, we can estimate the probability of entering metastability as:

$$p(metastability\ per\ \sec) = T_W F_C F_D . \tag{2.1}$$

Here, F_C and F_D represent the sampling clock frequency and the input data changing frequency, respectively. T_W denotes the metastability window around the active clock edge, where the latch might be metastable due to the unstable input.

After entering metastability, a latch could consume a random and unbounded time to resolve it. Given a synchronization time of s, it will fail to exit from metastability with a probability of:

$$p(failure\ per\ \sec) = T_W F_C F_D \cdot e^{-s/\tau} . \tag{2.2}$$

In (2.2) τ is the resolution time constant, which is determined by the circuit design and the fabrication process of a latch. We see that the probability of a latch keeping meta-stable decays exponentially with the synchronization time of s. It turns out to be crucial to prevent metastable sampling results from impacting other circuits.

Synchronization circuits are designed so as to minimize, or even eliminate in some cases, the probability of outputting metastability. By allowing a synchronization time long enough, we can expect to resolve the metastability inside a synchronization circuit with a negligible failure probability. Based on (2.2), the mean time between failures (MTBF) of a synchronization circuit can be derived as [15]:

$$MTBF = e^{s/\tau} / T_W F_C F_D . \tag{2.3}$$

Above MTBF presents a measure of synchronization reliability. A synchronization circuit is often designed to provide a MTBF that is orders of magnitude longer than the device target life time. According to (2.3), the MTBF of a synchronization circuit is exponentially dependent on the following two parameters:

- Synchronization time s
  For most designs based on standard cells, adapting s might be the first choice of designers, if not the only one, for improving the synchronization reliability. Any arbitrarily large MTBF can be guaranteed, statistically speaking, by allowing a sufficient synchronization time. Unfortunately, as shown in (2.2), this solution cannot eliminate synchronization failures. Once entering metastability, there is a finite probability, for any s, within which the metastability fails to settle due to the unbounded resolution time.
- Resolution time constant τ
  Given the fabrication process, τ is constant for a synchronizing latch (ignoring drifts caused by PVT variations). It determines the intrinsic speed of a latch to resolve metastability. In the past τ was known to scale down nicely with process. Recent measurements, however, have indicated the degradation effect on it [16]. It is implied that a larger s will need to be reserved in future technology nodes to achieve a given MTBF.

2.2.2 Synchronization Overhead

Due to the synchronization, GALS design tends to introduce potential performance loss compared with the synchronous design. For instance, robust communication across clock domains relies on the sufficiency of synchronization time as aforementioned. It in fact gives rise to the latency of data transfer. Depending on the structures of data flow, large synchronization latency could further worsen the throughput of a GALS system, which is, however, crucial in many applications.

Synchronization circuits also consume extra area and power. The dynamic power of a synchronization circuit is determined by the frequency of transferring data across clock domains. It can be marginal if intensive inter-clock communications are avoided. As to the area, it often depends on the complexity of a synchronization circuit, which in turn relies on the performance requirements. Therefore there is often a tradeoff between overheads and performance for the design of synchronization circuits.

In principle, the costs of clock synchronization necessitate to explore and advance the GALS design optimization at the system level.

2.3 GALS Interface Circuits

To address the synchronization issues, three classes of GALS interface circuits are proposed in the literature: boundary synchronizers, two-clock FIFOs, and local clock stretching. In this section, representative interface designs are reviewed, with a focus on the fundamental schemes for reliable synchronization.

2.3.1 Boundary Synchronizers

Synchronization can be done by sampling an input signal by two cascaded flipflops on the clock border, as shown in Figure 2.1. Once metastability occurs in the first-stage flipflop, there is an entire clock period T reserved for resolving it, with a MTBF of (2.3) at $s = T$. As a penalty it leads to a synchronization latency of up to two clock cycles on each data transfer. Inserting more cascaded flipflops, as in the pipeline synchronization [17], can further improve the MTBF, at a cost of even worse data latency.

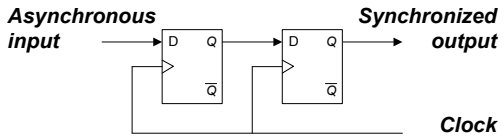


Figure 2.1 Cascaded double flipflops

More importantly, the input signal must be stable before being correctly sampled. It is guaranteed by the handshake operation between TX and RX. Figure 2.2 illustrates a typical boundary synchronizer, along with the corresponding handshake waveforms. The cycle time for transferring each of data items can be derived as:

$$T_D = k \cdot (l^{TX}_{SYNC} + l^{RX}_{SYNC} + d^{TX}_{FMS} + d^{RX}_{FMS} + d_{INT}). \quad (2.4)$$

There l^{TX}_{SYNC} and l^{RX}_{SYNC} are the synchronization latency of the double flipflops employed in the TX and RX clock domains, respectively. As mentioned, they can be at most two clock cycles. d^{TX}_{FSM} and d^{RX}_{FSM} denote the response delays of the TX and RX finite state machines (FSMs). The FSMs are required to support flow control on the channel. For a Moore FSM, a response delay of minimum one cycle is induced. In addition, d_{INT} is the round-trip interconnect delay of the handshake signals.

The parameter k in (2.4) is a handshake protocol determined factor of cycle time. Assuming that a four-phase protocol is employed on the channel, then we have $k = 2$. Four times of synchronization on the handshake signals are now involved in each data transfer. This often causes a prohibitively long cycle time of multiple TX and RX clock periods, and significantly limits data throughput. By avoiding the return-to-zero events, a two-phase handshaking protocol can halve the cycle time of each transfer, i.e., $k = 1$. This comes at the cost of an increased complexity in the TX and RX FSMs. Extra logic is required for conversion between the 4-phase (on the synchronous module side) and 2-phase (on the asynchronous channel side) handshake signals [18].

Also, the cycle time of a boundary synchronizer can be optimized by introducing concurrent handshaking on both sides. It is explored by the shared latch synchronizer, as shown in Figure 2.3 [19] [20]. The handshake loop between the TX and RX is split into two sub-loops, which are overlapped in time with each other. This leads to a cycle time of $k = 1$ in (2.4), the same as in the two-phase protocol case. In contrast, the handshake signals (*req*, *ack*, *req_sync*, and *ack_sync* in Figure 2.3) are kept four-phase level sensitive, thus facilitating the logic design of FSMs.

It is noteworthy that, instead of the data bus, the single-bit control signals (*req* and *ack* in above examples) are synchronized in a boundary synchronizer. Synchronizing a bus by a bank of double-flipflops might lead to a complete loss of data, if not nicely encoded, due to the undetermined resolution time of metastability.

2.3.2 Dual-Clock FIFOs

Due to the large cycle time consumed by each transaction, boundary synchronizers turn out to be applicable only for very low data-throughput communications. In order to improve the data rate, a circular FIFO timed by two independent clocks can be inserted in the channel for data buffering. Each data item is written at the TX clock into the cell addressed by the write pointer, and is read at the RX clock from the cell addressed by the read pointer. If the FIFO is neither full nor empty, continuous write and read can be done to sustain a high data throughput on the channel.

The main design concern of a circular FIFO is to prevent it from overflow (writing when full) and underflow (reading when empty). The status of a FIFO can be detected by comparing its write and read pointers. However, this introduces a challenge on safe synchronization of the write and read pointers (note that both are bus signals) across the clock domains. To avoid any spurious value occurring after synchronization, the FIFO pointers have to be encoded with a hamming distance of one between adjacent words. Since each time there is only a single bit changing, we can ensure that the synchronized pointer is either the current value or the previous one. Safe detection on the FIFO state is therefore achieved. In principle, the pointer coding schemes, adopted for addressing and synchronization respectively, determine the performance as well as the complexity of a dual-clock FIFO design.

Due to their high density, SRAMs are often used as the storage devices for large-size on-chip FIFOs. In this case, the binary coding is normally adopted for addressing. The corresponding Gray codes are then employed to synchronize the pointers crossing clock boundaries [21] [22]. The converting logic between binary and Gray codes, along with the full and empty states detection logic, usually limits the maximum frequency of a FIFO design. For the Gray codes with power-of-two length, the conversion can be done by simple XOR operations. Even-length Gray codes are also possible, but need more resources for (de)coding [23]. Figure 2.4 shows the block diagram of a typical SRAM-based dual-clock FIFO.

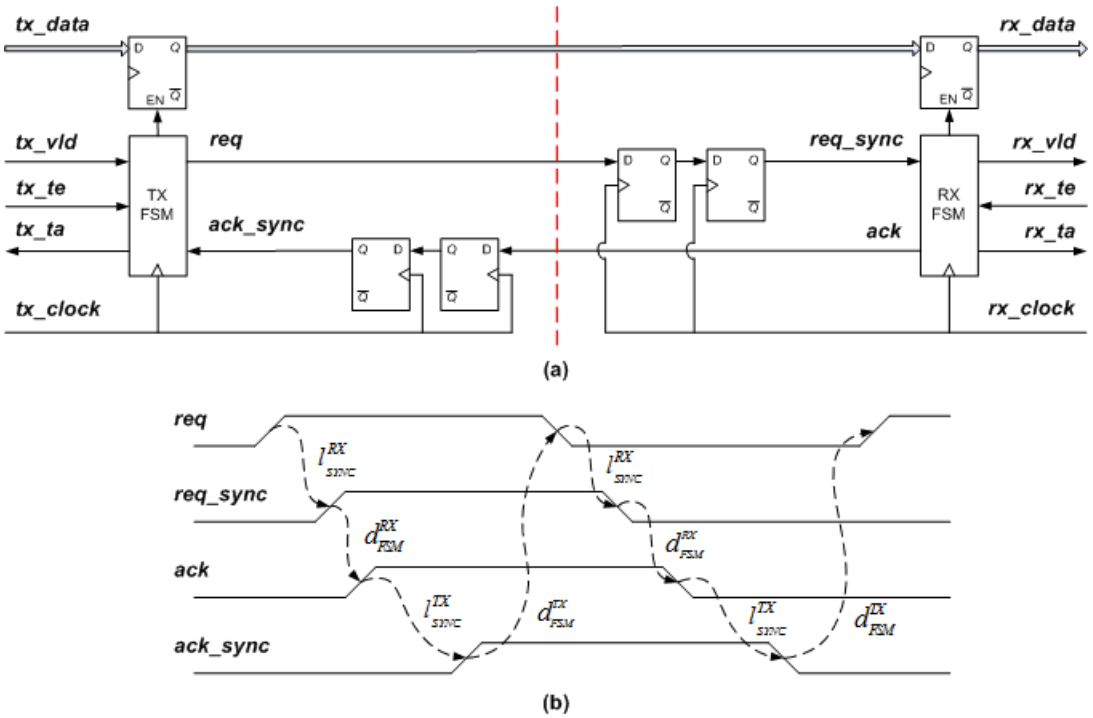


Figure 2.2 Typical boundary synchronizer: (a) structure and (b) signal waveforms

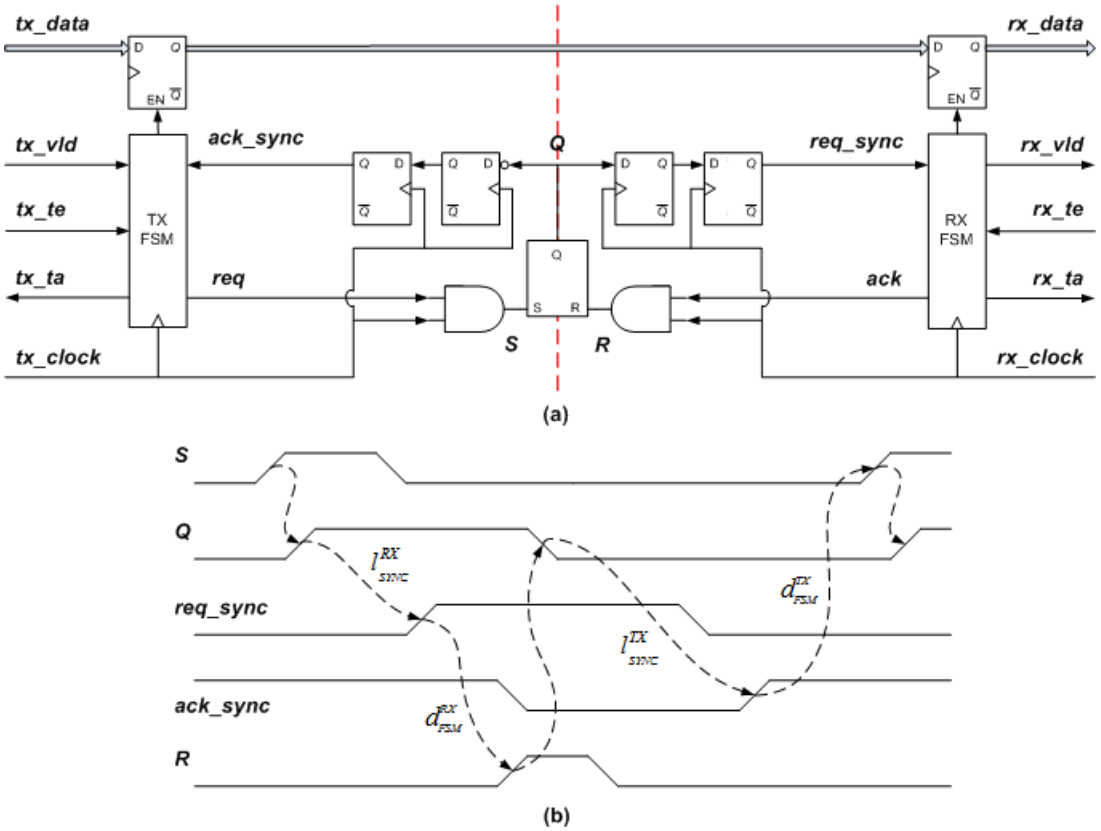


Figure 2.3 Shared latch synchronizer: (a) structure and (b) signal waveforms

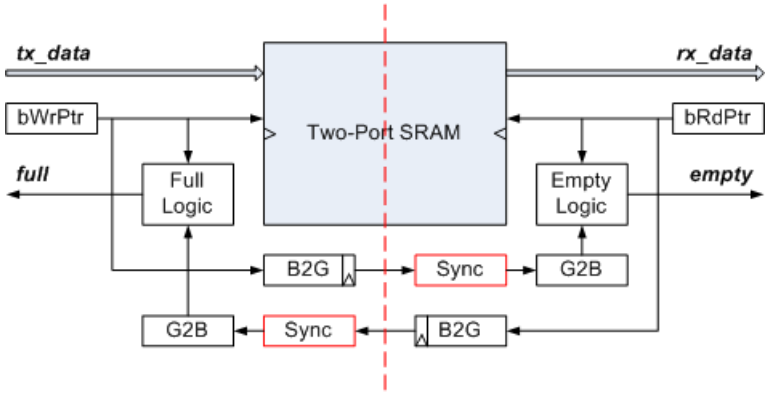


Figure 2.4 SRAM-based dual-clock FIFO design

For small-size FIFO designs, register arrays are preferred rather than SRAMs. As a result, the FIFO addressing relies on the one-hot coding of register enable signals. This feature is exploited to optimize the control logic of dual-clock FIFOs, as shown in [24] [25]. The one-hot FIFO pointers can be generated from circular shift registers, such as Bubble encoding [24] and Johnson encoding [25]. In general, these schemes benefit the dual-clock FIFO design in two aspects. First, the addressing codes inherently guarantee safe synchronization, and therefore no additional conversion is used. Second, any FIFO depth can be implemented, offering more flexibility for FIFO configuration (with small size) than the traditional Gray codes.

Special attention needs to be paid on the synchronization latency of FIFO pointers. It results in a "pessimistic detection" of the FIFO state: the FIFO is detected being full or empty for extra cycles after it exits from that state. Although the pessimistic detection doesn't corrupt FIFO data, it impacts the throughput. Each time the FIFO goes empty, the synchronization latency is imposed on resuming a new read access. Consequently, the throughput drop can be significant if a FIFO gets empty frequently. Supposing the FIFO is sufficiently deep, the synchronization latency can be hidden by the buffered data. However, this solution often leads to remarkable hardware overheads. More advanced dual-clock FIFO design, with low latency and small area as presented in [26], is based on the customized design of the full and empty detecting logic, therefore giving rise to much more design effort.

2.3.3 Local Clock Stretching

When applying boundary synchronizers or dual-clock FIFOs, a timing assumption is actually inferred: the sampling clock is pre-defined and rigid. Significant work also has been done to relax the constraint of fixed clock for safe synchronization. Basically, a ring oscillator is employed to generate the local clock. Its feedback loop, however, is gated by the requests from asynchronous channels. As a result, each clock cycle can be dynamically adapted according to the arrival time of input data. More precisely, instead of synchronizing the inputs to the clock, we are synchronizing the clock, if necessary, to the inputs. This represents the concept of local clock stretching [27] [28].

A. Two Clocking Strategies

As depicted in Figure. 2.5, there exist two schemes in the literature to implement clock stretching: data driven and pausible clocking. In both cases, a Muller C-element is inserted in the feedback path of a ring oscillator for clock gating. Depending on the gating input of the Muller C-element, however, the clocking strategies employed in the two schemes can be quite different from each other [29].

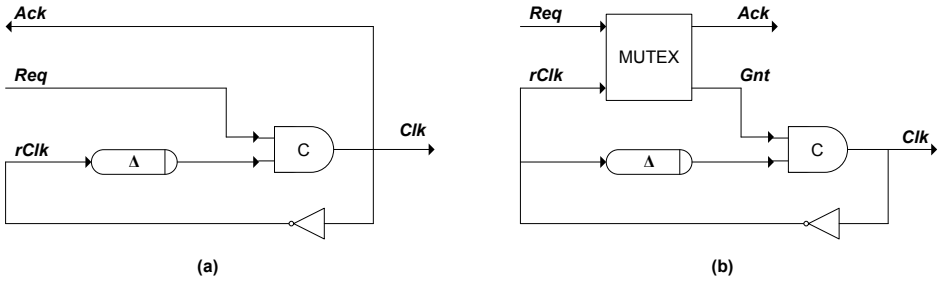


Figure 2.5 Two clocking strategies: (a) data-driven clocking and (b) Pausible clocking

For the data-driven clocking, the output clock is gated by the four-phase handshake signal *req* from the channel. Every time receiving a valid input, *req* is asserted and a single clock pulse is then generated. Since the clock is always quiescent when the input arrives, it prevents metastability altogether from occurring. Further, dynamic power is saved by avoiding switching activity when no data is ready for processing. Data-driven clocking is most suitable for single-stage designs. For unrolled pipeline circuits, extra flushing mechanisms often have to be implemented. An example of data-driven design is reported in [30], where it is applied for memory access.

In contrast, the data request *req* has been arbitrated with the clock request *rClk*, by a MUTEX element, in the pausible clocking scheme. At any moment, only one request is allowed by the MUTEX, thus ensuring the temporal separation of the clock rising edges from the data transitions. Synchronization failure is eliminated as a result. However, it comes at the expense of potential metastability inside the MUTEX due to the concurrent requests from data and clock. An unbounded clock pausing can be caused to resolve the metastability. On the other side, the clock will be oscillating uninterruptedly if no data transfer is required. For most pipelined designs, this is desired in comparison with data-driven clocking. Clock gating also can be introduced to lower dynamic power in system sleep mode [31].

B. Pausible Clocking Scheme

Figure 2.6 depicts a typical GALS block based on pausible clocking. Surrounding synchronous functional modules is an asynchronous wrapper [32]. It times synchronous modules by a locally generated pausible clock and schedules data transfer across clock domains by a number of asynchronous input and output port controllers (IPC and OPC). The circuit design of the asynchronous wrapper is briefed in this subsection.

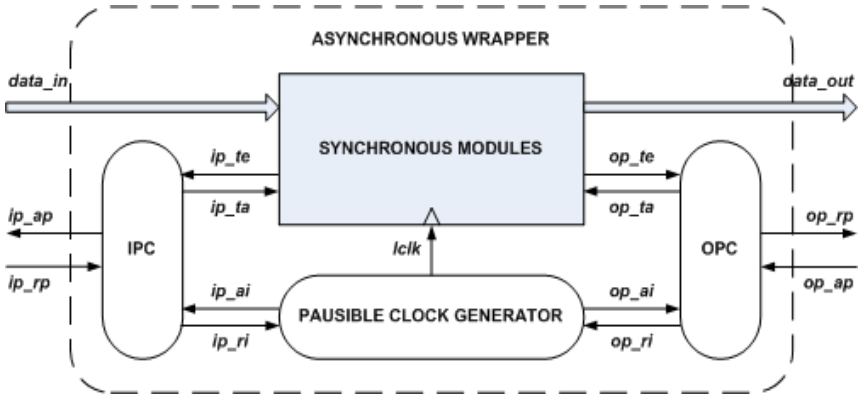


Figure 2.6 A GALS block based on pausible clocking

As shown in Figure 2.7, the practical design of a pausible clock generator evolves from its prototype circuit of Figure 2.5 (b), with two aspects of optimization. First, an arbiter bank which consists parallel of a number of MUTEX elements, is employed for clock stretching [33]. Each MUTEX corresponds to a request ri from an IPC or OPC. Attributed to the parallel arbitration, several ri can be acknowledged in a single clock cycle simultaneously. Therefore, concurrent communications via multiple ports can be supported. This is a significant advantage in comparison with the arbiter trees [27]. In addition, the parallel arbitration provides scalability to any number of input and output ports accommodated by an asynchronous wrapper.

Second, a programmable delay line is applied in the ring oscillator to on-chip adjust the clock frequency. The key concern is to obtain a high tuning precision with low area and power overheads of the delay line. All-digital delay lines can be constructed by a number of identical delay slices using standard cells [34] [35]. Its delay is adjusted by programming the amount of slices activated in the loop. Sub-gate delay resolution is also possible by custom design of delay slices [36]. As it is well known, the frequency of a ring oscillator is very sensitive to PVT variations. This necessitates real-time dynamic calibration of the delay line, for example, against a low frequency reference clock [34] [37]. Nevertheless, due to undeterministic stretching of clock cycles, the pausible clock generator generally cannot guarantee a fixed clock frequency.

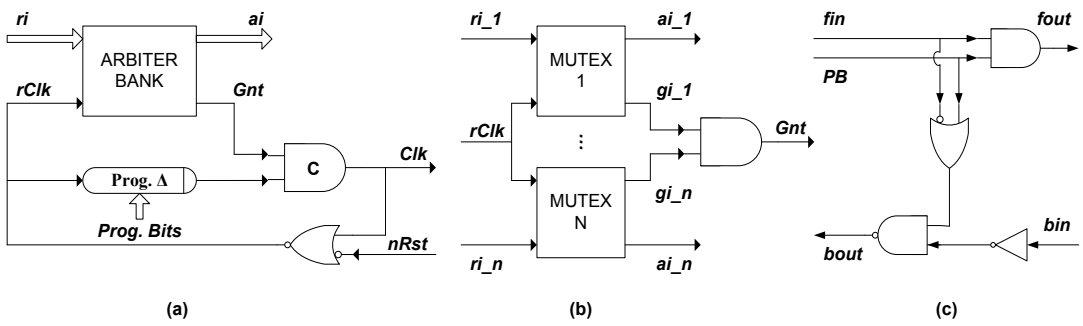


Figure 2.7 Pausible clock generator: (a) block diagram, (b) arbiter bank and (c) delay slice

In the pausible clocking scheme, each data item is transferred via a point-to-point channel, which consists of a pair of I/O port controllers, and if necessary, a data buffer between them. Three primary operations can be characterized for a port controller. First, it needs to synchronize the clock phase to the data arrival time. Second, it is responsible for handshaking over a channel with its partner port. Third, it has to communicate with the functional modules to support flow control. Accordingly, each port controller has been equipped with three handshake interfaces as seen in Figure 2.6.

To obtain a high data rate the port controllers need to act independently of the local clock. This is achieved by implementing them using asynchronous FSMs. Instead of the clock edges in a synchronous FSM, an asynchronous FSM is triggered by the transitions on its inputs and feedback outputs. When obeying the input-output mode, its behavior can be specified by the I/O signal transition graphs (STGs). Some academic tools, such as *Petrify*, have been developed to synthesize the gate-level netlist of an asynchronous FSM based on its STG specifications [38]. The *Petrify* derives speed-independent (SI) logic: it is hazard-free under the condition of iscochronic forks. This necessitates the hard/soft macro design of the port controllers to constrain the interconnect delays.

Muttersbach *et al*. introduced two families of I/O port controllers, namely demand-type and poll-type ports, to address diverse conditions for clock gating [39]. A demand-type port assumes that each data transfer needs to be done before any further processing takes place. It stops the local clock as soon as getting enabled, regardless whether the channel is ready for an immediate transfer. This in fact orthogonalizes computation and communication of a GALS block. The states of functional modules are frozen by clock gating throughout each data transfer. In contrast a poll-type port allows continuous data processing with transfer pending. Once being activated, it waits until the channel being valid to stop the clock. By this means, it attempts to retain, as much as possible, the working speed of a GALS block.

Attention is deserved by the orthogonalization property of the demand-type ports. Actually, it facilitates the flow control by a port controller: it is implicitly done by clock gating. No transfer acknowledge signal (*ip(op)_ta* in Figure 2.6) is required to indicate the transfer status, and the flow control logic even can be fully saved for the functional modules in many cases. As a penalty, however, the demand-type ports could introduce frequent clock pauses, impacting the system performance consequently. Intensive inter-clock communication will further exacerbate this issue. The demand-type ports indeed impose a challenge on system design to mitigate the performance loss.

For reference, Figure 2.8 depicts an alternative design of a pausible clock generator, named the stretchable clock in [32]. There the stretch signal is asserted synchronously but released asynchronously to the clock. It essentially addresses a particular application of pausible clocking by exclusively using the demand-type ports. Considering the determined arrival time of requests (*ri*) relative to the local clock in this scenario, the arbiter bank in Figure 2.7 has been replaced by a simple AND gate.

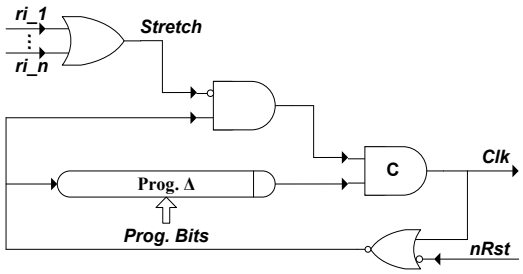


Figure 2.8 Bormann's stretchable clock [32]

2.4 Comparison of GALS Interface Designs

Table II.2 gives a comparison between typical GALS interface designs in terms of major synchronization metrics. The pausible clocking scheme outperforms the others if concerning the synchronization reliability. By the dynamical adjustment of clock phase, it allows the value-safe synchronization, i.e. eliminating synchronization failures from GALS designs. Boundary synchronizers and dual-clock FIFOs are both time-safe. That is, a negligible but non-zero probability of synchronization failure, measured by MTBF, can be achieved by reserving a sufficient synchronization time.

Boundary synchronizations suffer the most from the data throughput drop, due to the synchronization latency of double-flipflops. Employing a dual-clock FIFO can even worsen the data latency, because each newly written data has to wait to be read until all the early buffered data are consumed. But it contributes to sustain the data throughput, given the FIFO is kept neither empty nor full. A sufficiently large FIFO is required for this reason, often leading to a remarkable overhead in silicon area.

Quite significant design efforts are needed for the pausible clocking based GALS design. First, the asynchronous port controllers are fundamental different from the synchronous FSMs in behavioral modeling, logic synthesis and timing analysis. Second, the absence of Muller C-element, MUTEX and delay lines from most standard cell libraries challenges the practical implementation of pausible clock generators. Third, as a result, the hierarchical layout is required to achieve system-level timing closure.

Table II.2 Comparison of GALS interface designs

	Reliability	**Latency**	**Throughput**	**Area Overhead**	**Design Effort**
Boundary Synchronizers	Time safe	Medium - High	Low	Low	Low
Two-Clock FIFOs	Time safe	High	High	Medium - High	Medium
Pausible Clocking	Value safe	Low	Medium - High	Medium	High

2.5 Summary

Interface circuit design imposes a fundamental challenge on the implementation of GALS systems. Synchronization reliability, overheads on throughput, area and power, as well as design effort, are the major concerns. Boundary synchronization, dual-clock FIFO and pausible clocking represent the typical solutions proposed in the literature for fully asynchronous communication.

Today, most of the GALS systems are actually based on the dual-clock FIFOs. Although pausible clocking promises an alternative to address the synchronization issues more elegantly and efficiently, it is still considered by the industry as a niche technique. The existing schemes suffer from some critical problems for the practical applications. In the following chapters, our recent progress in the advanced GALS design based on pausible clocking will be elaborated.

Chapter 3

Synchronization by Mutually-Exclusive Latching

As addressed in Chapter 2, safe synchronization by pausible clocking relies on the dynamic scheduling of clock switching activity. This actually implies an assumption of negligible clock propagation delay from the source node to leaf registers. Unfortunately, it is not always the case in reality. The propagation delays throughout the on-chip clock distribution networks often cannot to be ignored. As revealed by [40], taking the clock-tree delays into account would fundamentally challenge the traditional synchronization schemes of pausible clocking.

This chapter begins with an in-depth analysis of the synchronization failures in the pausible clocking schemes. Then representative solutions reported in the literature are reviewed. An optimized synchronization interface based on pausible clocking, named Mutually-Exclusive (ME) Latching, is proposed. It differs from the traditional schemes in two aspects: the mutually exclusive latching on input and the acknowledge-window configurable clock generator. Synchronization reliability of the ME latching, taking into consideration the clock-tree delays, is further elaborated.

3.1 Synchronization Failure due to Clock-Tree Delay

3.1.1 MUTEX Acknowledge Window

An implementation of the MUTEX, along with the I/O signals used in the pausible clock generator, is depicted in Figure 3.1 [41] [42]. It is a pair of cross coupled NAND gates, followed by a metastability filter. At any moment, only one high input request can be acknowledged, on a first-come-first-serve basis. If both inputs are asserted concurrently, the NAND gates get metastable, but the MUTEX outputs are kept low by the filter circuit. Until the metastability is eventually resolved the MUTEX will randomly select one request to be acknowledged. It works as an arbiter between the requests from the ring oscillator (*rClk*) and port controllers (*ri*) in the clock generator.

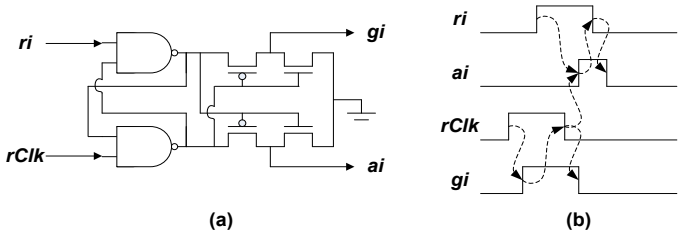


Figure 3.1 MUTEX: (a) Seitz structure and (b) I/O behavior

We define the MUTEX acknowledge window w_{MUTEX} to represent the duration in a clock cycle when a *ri*+ from the port controllers can be acknowledged. According to the mutual exclusion property on the MUTEX outputs, w_{MUTEX} is basically determined by the off-phase of *rClk*:

$$w^{N}_{MUTEX} = \{NT \le t < (N+1)T \mid rClk(t) = Low\}. \quad (3.1)$$

Therefore, by adjusting the duty cycle on *rClk*, the width of w_{MUTEX} can be configured accordingly. As illustrated in Figure 3.2, it is even possible, when necessary, to have a w_{MUTEX} varying from one cycle to another. Nevertheless, in most cases *rClk* is simply an inverted version of the output clock signal. This leads to a constant width of w_{MUTEX}, usually being half a clock period.

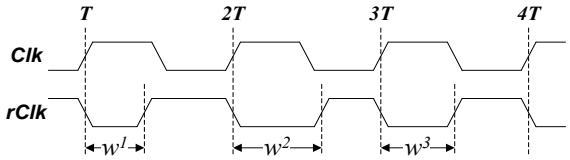


Figure 3.2 MUTEX acknowledge window

Consider the particular situation, where *ri*+ happens at the transition time of *rClk*. A *ri*+ arriving at the same moment with *rClk*- will be acknowledged by the MUTEX immediately. In contrast, a *ri*+ coming simultaneously with *rClk*+ will introduce metastability in the MUTEX. It might get acknowledged, at a probability of 50%, in an indeterminate and unbounded resolution time, as addressed in Section 2.2.1. Without the loss of generality, we have ignored the transition time of *rClk* from the above definition of w_{MUTEX}. It is worth to mention that there is no constraint that can be imposed on the occurrence time of *ai*+. In case that the MUTEX gets into metastability, *ai*+ can be set in an arbitrary delay, theoretically speaking, because of the random resolution time. This is the reason for us to specify w_{MUTEX} regarding *ri* rather than *ai*.

Essentially, the definition of w_{MUTEX} lays a foundation of the performance analysis of GALS design by pausible clocking. Any *ri*+ falling in w_{MUTEX} will be acknowledged immediately by the MUTEX, just in a propagation delay of several combinational gates. This accounts for a negligible latency to transfer a data item across clock boundaries. In contrast, a *ri*+ occurring outside w_{MUTEX} of the current clock cycle has to wait until the arrival of w_{MUTEX} in the next cycle for being acknowledged. This imposes a remarkable overhead on data latency. From the perspective of data throughput, it is thus preferred to have each *ri*+ falling inside the w_{MUTEX}. Due to asynchronous communication, however, the arrival time of *ri*+ is often randomly distributed in time to some extent. This challenges the high performance GALS design based on pausible clocking. Behavioral optimization of the port controllers turns out to be crucial. Clock distribution delays and port interconnect delays further complicate the design and analysis. A systematic study on these issues will be presented in Chapter 4.

3.1.2 Synchronization Failure

A typical synchronization interface based on pausible clocking is shown in Figure 3.3 [39]. Input data from the transmitter are first latched by the datapath latches *L*. The latched data are further sampled by the input flipflops *FF* in the receiver. *L* is normally used to decouple the I/O port controllers for high-throughput communication on a data link. When receiving an input data from the transmitter, a *ri*+ is set by the IPC asking for a clock pause. As soon as *ri*+ gets acknowledged by the MUTEX, *ai* is asserted to load the data into *L*, and at the same time *gi* is kept low to suspend the clock in the off phase. Once the data is stably latched into *L*, which is guaranteed by a high pulse on *ai* with sufficient width, *ri* is de-asserted by the IPC, allowing a *sClk*+ to sample the data safely by *FF*. Synchronization interfaces based on pausible clocking that are reported in the literature are mostly variations of Figure 3.3 [40].

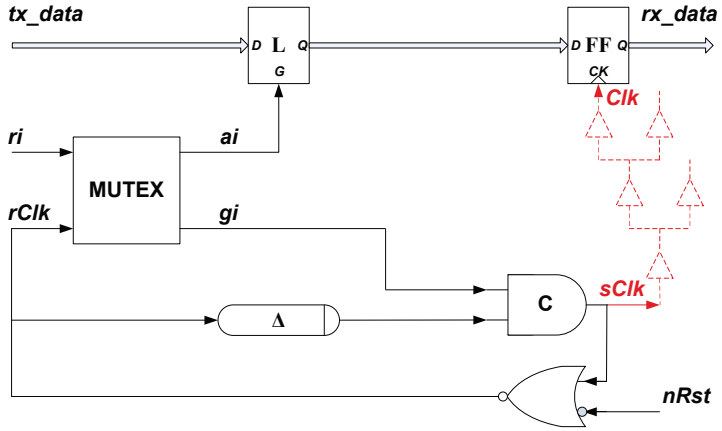


Figure 3.3 Typical synchronization interface based on pausible clocking

In practice, the clock signal always suffers a propagation delay d_{CT} from the source node *sClk* to leaf pins *Clk* on *FF*. It is introduced by the clock distribution networks, usually being the clock buffer trees, inserted on the chip for driving the large clock fan-out of numerous registers. Figure 3.3 exemplifies partially a clock tree in the red dashes. Unfortunately, taking into account the propagation delays of on-chip clock trees nullify above timing conditions for safe synchronization. First of all, given a clock tree with a multi-cycle delay, i.e., $d_{CT} > T$, which is often the case for high-speed systems, halting *sClk* cannot prevent *FF* from synchronization failures, because data are sampled at the free-running clock edges buffered in the clock tree. Even for a sub-cycle clock delay of $d_{CT} < T$, it cannot achieve the safe synchronization, as discussed below.

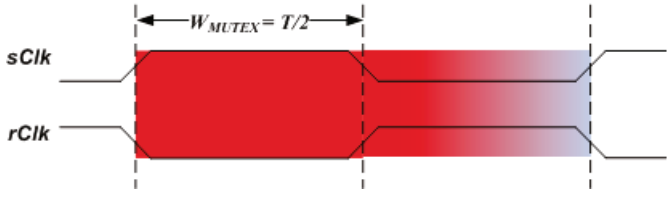


Figure 3.4 MUTEX acknowledge window in Figure 3.3

The synchronization interface in Figure 3.3 introduces an *rClk*, which is simply the inverted version of the output clock signal *sClk*. Ignoring the short propagation delay of the NOR gate, its w_{MUTEX} can be approximated by the on-phase of *sClk* in each cycle, as drawn in Figure 3.4. Inside the w_{MUTEX}, *ai* can be asserted by the MUTEX, as soon as it receives a *ri*+ from the asynchronous port controllers. By contrast, a *ri*+ appearing at the same time with *sClk*- will lead the MUTEX into metastability. In this particular scenario, *ai* might become high in an arbitrary delay, as aforementioned, due to the indeterminate and unbounded resolution time of metastability. This causes a fact that the input data could be latched into *L* at any moment within the entire clock cycle. Under this condition, safe sampling of *FF*, which is timed by the *Clk* is allowed only when d_{CT} is zero. Any delay from *sClk* to *Clk*, even with $d_{CT} < T$, might lead *FF* to be metastable, due to timing violations at sampling the latched data.

It is worth to estimate the failure rate in the synchronization interface of Figure 3.3. The chance of getting synchronization failures of *FF* is determined by the occurrence probability of *ai*+ within the metastability window surrounding the sampling edges of *Clk*+. Because of asynchronous communications, typically the arrival time of *ri*+ can be modeled by a uniform distribution over the clock cycle. This gives rise to a constant probability density of *ai*+ in the w_{MUTEX}. Thus, when *Clk*+ falls in the w_{MUTEX}, that is, $0 \le d_{CT} < T/2$, the probability of synchronization failures can be obtained by (2.1). It is rewritten in (3.2) for the convenience of discussion, where T_W, F_C and F_D represent the width of the metastability window of *FF*, the frequency of the sampling clock *Clk* and the changing rate of the input data, indicated by *ri*+, respectively.

$$p(metastability\ per\ \sec) = T_W F_C F_D . \tag{3.2}$$

By contrast, *ai*+ occurring outside the w_{MUTEX} comes from the resolution of metastability by the MUTEX. According to (2.2), given two synchronization times of s and $s + \Delta s$, the probability for the MUTEX to assert *ai* during the time slot of $s < t < s + \Delta s$ can be derived as (3.3). The coefficient of 1/2 accounts for the unbiased outputs of *ai*+ and *gi*+, when the MUTEX eventually exits from metastability.

$$p(ai+\ per\ \sec) = \frac{1}{2} T_W F_C F_D \cdot e^{-s/\tau}(1 - e^{-\Delta s/\tau}) . \tag{3.3}$$

Substituting $\Delta s = T_W$ in (3.3), we get the occurrence probability of synchronization failures on *FF*, given a d_{CT} satisfying that $T/2 \le d_{CT} = T/2 + s < T$, as presented in (3.4). It decays exponentially with the synchronization time *s*, which is determined in fact by the delay of *Clk*+ relative to *sClk*-, ignoring the NOR gate delay. Taking (3.2) and (3.4) both into account, the probability of synchronization failures in each clock cycle can be graphically indicated by the red shadow in Figure 3.4. The darker the color, the higher is the probability. There is no safe time region available for *FF* to sample the input data without the risk of causing synchronization failures.

$$p(metastability\ per\ \sec) = \frac{1}{2} T_W F_C F_D \cdot e^{-s/\tau}(1 - e^{-T_W/\tau}) . \tag{3.4}$$

3.2 Literature Review on Safe Synchronization

Synchronization failures caused by clock-tree delays fundamentally challenge the GALS design based on pausible clocking. In this section representative solutions in the literature for safe synchronization are reviewed. Note that, similar to pausible clocking schemes, data driven clocking suffers from synchronization failures as well [29].

The metastability-free, i.e., value-safe, synchronization based on pausible clocking can be achieved by avoiding simultaneous requests from port controllers (*ri*+) and the ring oscillator (*rClk*+). A simple approach is to implement I/O ports in a GALS system exclusively in the demand type. *ri*+ from the demand-type ports are triggered by *Clk*+, hence occurring in a determinate delay relative to *rClk*+. Consequently, the MUTEX is not required for arbitration, and the clock generator can be simplified to be the stretchable clock of Figure 2.8. Due to the timing constraints on the demand-type ports, such a design style is applicable only for sub-cycle clock-tree delays. Also, it can induce data-throughput loss, as mentioned in Section 2.3.3, by intensive clock pauses. Nevertheless, to the best of our knowledge, it seems the only value-safe scheme by pausible clocking until now. There are plenty of alternative "metastability free" solutions in the literature. Many rely on the Muller C-element as an arbiter instead of the MUTEX. Unfortunately, the authors ignore an important fact: the Muller C-element suffers from metastability as well, once the inputs switch simultaneously in opposite directions or fail to keep stable before being responded [43] [44].

Generally speaking, it is possible to build a ring oscillator out of clock buffer networks. In this case, clock-tree delays can be efficiently absorbed by the clock generator. Under the condition of $d_{CT} < T/2$, the clock insertion delay will be fully hidden, thus permitting the value-safe synchronization by pausible clocking. Integrating clock trees in the ring oscillator was first addressed in [45]. Actually, the authors found their work on the stretchable clock. More important, the proposed scheme fits for linear pipelines only, and could hardly be generalized for other architectures.

As a contrast, the metastability-negligible, i.e., time-safe, synchronization based on pausible clocking was studied in depth by R. Dokbin *et al*. in [40] [46]. It is shown that negligible probability of synchronization failures can be expected, if generating clock trees under certain delay constraints. Given a clock period, the allowed time regions for clock-tree insertion depend on the synchronization time to be reserved for achieving a target MTBF. A novel synchronization scheme, named locally delayed latching (LDL), was further proposed by the authors. Rather than in the clock generator, it performs the arbitration at sampling registers using MUTEXs. No restriction on clock-tree delays for input synchronization is required as a result. On the other side, however, the clock will never be suspended during arbitration in the LDL scheme. To afford sufficient time for the MUTEXs settling from metastability, strict delay constraints are imposed on the input port controllers and the combinational logic following the sampling registers. This indeed affects the practical applications of the LDL scheme.

3.3 Synchronization by Mutually-Exclusive Latching

An optimized synchronization interface based on pausible clocking, which is named mutually-exclusive (ME) latching, is presented in Figure 3.5. As highlighted in green, it differs from the traditional schemes in two aspects: the mutually exclusive latching on input data, and the clock generator with configurable acknowledge window. Both contribute to reliable synchronization.

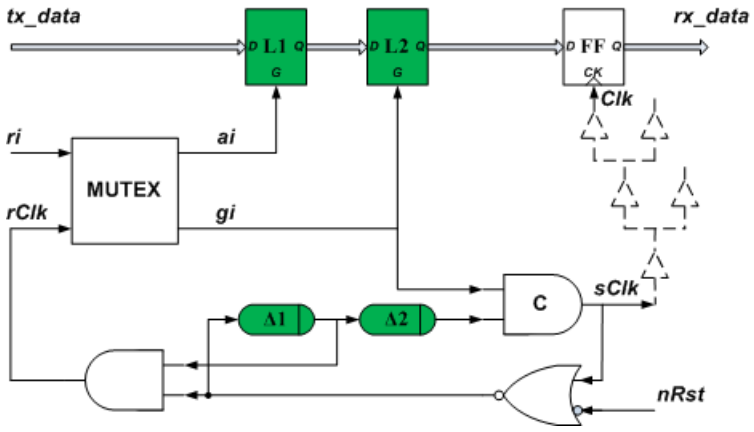


Figure 3.5 Synchronization by Mutually-Exclusive Latching

3.3.1 Mutually Exclusive Latching on Input Data

Two cascaded stages of data-path latches, rather than the single one in Figure 3.3, are employed here for input data buffering: *L1* latches the data from the transmitter, and *L2* feeds the data to the receiver. In particular, *L1* and *L2* are gated by the output signals *ai* and *gi* of the MUTEX, respectively. Due to the mutual exclusion between *ai = 1* and *gi = 1*, *L1* and *L2* are orthogonal, i.e., never overlapped, in time for data latching, hence the name. It leads to an efficient isolation between the data sampling in the receiver and the data loading from the transmitter. As analyzed below, this greatly improves the data synchronization by pausible clocking.

During the off-phase of *rClk*, *gi* keeps low and *L2* is stably locked. Consequently, any *Clk+* arriving at *FF* in the meantime can sample the data safely. By contrast, once *rClk* turns to be high, *gi* could be asserted actually at any moment to load data into *L2*, due to the indeterminate resolution time of metastability. As a result, any *Clk+* within the on-phase of *rClk* can cause synchronization failures of *FF*. Taking both scenarios into account, we see that in each clock cycle a safe time region is provided for sampling data by *FF*, which is in fact the off-phase of *rClk*. According to (3.1), it turns out to be the MUTEX acknowledge window. In other words, we are able to avoid synchronization failures by the ME latching, if distributing *Clk+* inside the w_{MUTEX}. This accounts for a fundamental advantage over the existing interface designs based on pausible clocking, in which no safe time region exists for input data synchronization.

3.3.2 Clock Generator with Configurable Acknowledge Window

Special attention is deserved to optimize the w_{MUTEX} for ME latching. The optimal width of w_{MUTEX} can be different, depending on a variety of design conditions. A clock generator, which allows configurability of acknowledge window, is desired for this reason. Figure 3.5 presents a simple solution. Two cascaded delay lines Δ1 and Δ2, instead of the single one in Figure 3.3, are employed to construct the ring oscillator. Both can be programmed in delay, under the following constraint of (3.5), to maintain a normal clock period of *T* when no clock pause happens:

$$d_{\Delta 1} + d_{\Delta 2} = \frac{T}{2} - (d_C + d_{NOR}). \tag{3.5}$$

Furthermore, a delay feedback loop is introduced for generating *rClk*. It is asserted after both, the NOR gate and Δ1, are output high, but is de-asserted once the NOR gate turns low. The off-phase duration of *rClk*, and also the width of w_{MUTEX}, can be derived:

$$w_{MUTEX} = 2d_{\Delta 1} + d_{\Delta 2} + (d_C + d_{NOR}). \tag{3.6}$$

Taking (3.5) into account, above expression can be simplified as:

$$w_{MUTEX} = T - d_{\Delta 2} - (d_C + d_{NOR}) \approx T - d_{\Delta 2}. \tag{3.7}$$

It is shown that, ignoring the short propagation delays of the Muller C-element and the NOR gate, the width of w_{MUTEX} in Figure 3.5 is determined by the delay of Δ2. Therefore, by adjusting the length of Δ2, it can be configured accordingly. Figure 3.6 shows an example of the w_{MUTEX} in the ME latching synchronization. The safe region offered for sampling data by *FF* is highlighted by the green shadow. In the next section we will derive the optimal configuration of w_{MUTEX} in various design conditions.

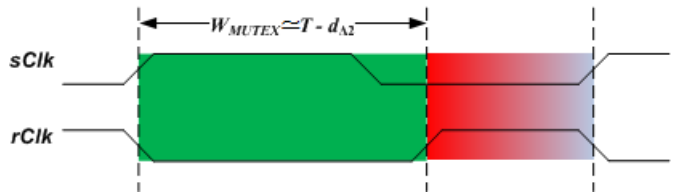


Figure 3.6 Acknowledge window in ME latching synchronization

3.3.3 Timing Assumptions

Above analysis disregards the propagation delays on *ai* and *gi* from the MUTEX to the latches. Because both signals are used to enable the latches, clock-tree synthesis can be performed on them. For two reasons, the delays caused by the *ai* and *gi* buffer trees are negligible in practice. First, the fanout of both signals is pretty low in most cases: it is determined by the datapath width, rather than the design complexity. Given a 64-bit data bus for instance, only three stages of buffers are needed, at a typical fanout ration of 4. Second, the skew constraints on both signals can be sufficiently relaxed, as there is no combinational logic timed by them. Detailed timing analysis on the ME latching scheme, in the context of GALS data-link design, will be addressed in Chapter 5.

3.4 Optimization of Acknowledge Window

It is shown that, to avoid synchronization failures of *FF* in the ME latching scheme, the sampling clock edges *Clk+* should always occur in the safe region, i.e., within the acknowledge window. As depicted in Figure 3.6, the MUTEX acknowledge window is actually aligned with the source clock edges *sClk+* in each of clock cycles. Once *sClk* is suspended for a clock pause, the safe region will be shifted as well. This introduces an uncertainty on the position of the safe time region for synchronization. Taking it into account, below we will explore the optimal designs of ME latching.

3.4.1 Timing Constraint on Fractional Clock-Tree Delay

For the convenience of analysis, we can express the clock-tree delay d_{CT} by (3.8), where N denotes the maximum number of clock cycles that may be buffered in a clock tree at one time, and Δ_{CT} the fractional clock-tree delay satisfying $0 \leq \Delta_{CT} < T$. If $N = 0$, (3.8) accounts for the sub-cycle delays on clock trees. Otherwise it addresses the case of multi-cycle clock-tree delays.

$$d_{CT} = NT + \Delta_{CT}\,,\ N = 0,\ 1,\ 2,\ \ldots\ . \tag{3.8}$$

Assume that there is no pause on *sClk*. Under this condition, the timing constraint for safe synchronization by the ME latching is specified as (3.9). That is, the fractional clock-tree delay, regardless of the value of N in (3.8), has to be smaller than the width of the MUTEX acknowledge window.

$$\Delta_{CT} < w_{MUTEX}\,. \tag{3.9}$$

According to (3.9), a wide acknowledge window is desired for clock tree synthesis. However, when the clock pause is taken into consideration, additional constraints can be introduced on the acknowledge window. A main issue is to tolerate the position uncertainty of the safe time region in data sampling. Depending on the value of N in (3.8), two scenarios can be distinguished as follows.

3.4.2 Value-Safe Synchronization for Sub-Cycle Clock-Tree Delay

Consider that $N = 0$, i.e., the sub-cycle delays on the clock trees. Figure 3.7 depicts an example of the clock pause in this case. As can be seen, *sClk* gets a suspension for a while in the second cycle: its off-phase is stretched and the next rising edge is delayed.

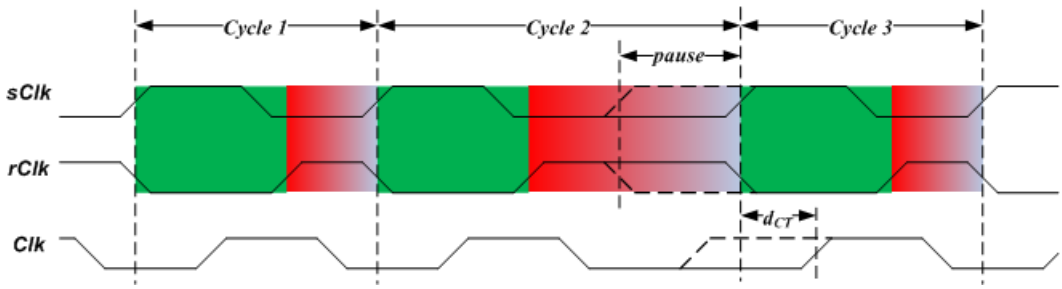


Figure 3.7 Clock pause at sub-cycle clock-tree delay

In Figure 3.7, the delayed *sClk+* of the third cycle leads to a shift of the safe time region (the green shadow). However, because $N = 0$, there is no rising edge buffered in the clock tree. As long as *sClk+* is gated, *Clk* is suspended in its off phase. Once *sClk* is asserted, the third clock cycle gets into the safe time region immediately. On the other side, *Clk* becomes high in a delay of d_{CT} to sample the input data. As a result, *Clk+* is shifted exactly in accordance with the safe region. In other words, clock pauses turn out to have no impact on data sampling for a sub-cycle clock-tree delay. The constraint of (3.9) therefore can be directly applied with $\Delta_{CT} = d_{CT}$.

To allow sufficient flexibility for the synthesis of clock trees, it is thus preferred to have a wider acknowledge window. According to (3.7), the w_{MUTEX} in the ME latching scheme can be maximized when the delay on Δ2 is configured to be zero. This in fact leads to the bypass of Δ2 in the ring oscillator, as shown in Figure 3.8.

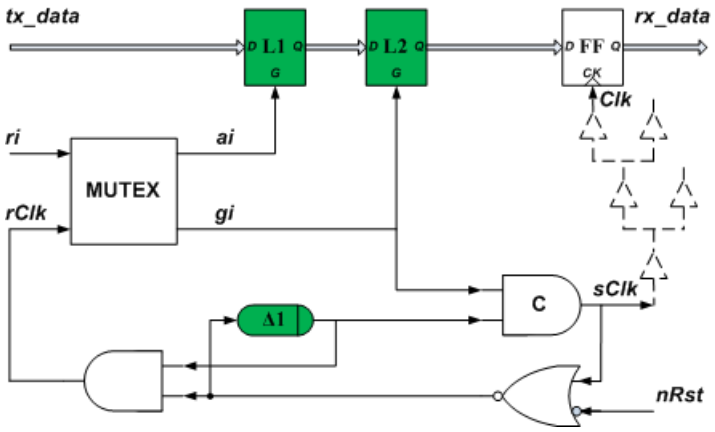


Figure 3.8 ME latching synchronization with Δ2 bypassed

The width of w_{MUTEX} in this case can be derived as (3.10). Given a clock period, it depends on the delays of four simple gates that are involved in the ring oscillator.

$$w_{MUTEX} = T - (d^0_{MUTEX} + d_C + d_{NOR} + d_{AND}) . \tag{3.10}$$

In (3.10) d^0_{MUTEX} is the intrinsic propagation delay of a MUTEX when no metastability is caused. Actually, it is the NAND gate delay in Figure 3.1. Compared with the clock period, the sum of above gate delays is typically marginal. For instance, it accounts for approx. ten fanout-of-4 (FO4) inverter delays, when synthesized using the IHP 130-nm CMOS technology. Consequently, Figure 3.8 often results in a MUTEX acknowledge window, which is also the safe time region for data synchronization, dominating the entire clock cycle, as illustrated below.

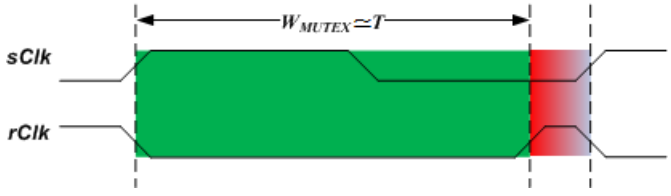


Figure 3.9 Maximum acknowledge window in ME latching synchronization

Ignoring the short gate delays in (3.10), we further obtain:

$$w_{MUTEX} \approx T \,. \tag{3.11}$$

That is, by bypassing Δ2 in the ring oscillator the ME latching synchronization offers a w_{MUTEX} which reaches its maximum width of being the clock period. Almost the whole clock cycle is safe for *FF* to sample the input data without introducing metastability. In other words, the full flexibility is reserved for the clock-tree synthesis in the layout. This way we achieve the optimal design of the value-safe synchronization, in the condition of the sub-cycle clock-tree delay.

3.4.3 Time-Safe Synchronization for Multi-Cycle Clock-Tree Delay

In contrast, consider the scenario of $N \geq 1$, i.e., the multi-cycle delays on the clock buffer trees. As aforementioned, once *sClk* is suspended for a clock pause, the safe time region in the next cycle will be shifted accordingly. However, there are *N* rising edges still buffered in the clock tree, which will arrive at *FF* without any extra delay. It is of importance to note that the potential clock pause on *sClk* is uncorrelated with the clock-tree insertion delay. As a result, *FF* might be triggered outside the safe region to sample the input data, leading to metastability. This means, in case of the multi-cycle clock-tree delay, we cannot avoid synchronization failures by the ME latching. A synchronization failure, introduced by the clock pause with a clock-tree delay at $N = 1$, is presented in Figure 3.10 for example.

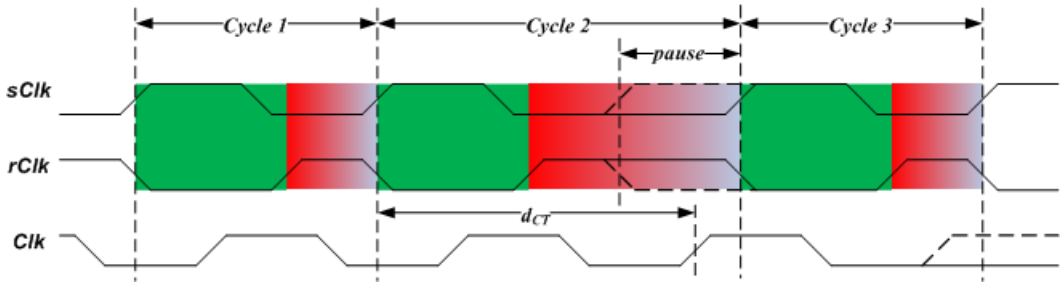


Figure 3.10 Clock pause at multi-cycle clock-tree delay

However, the time-safe synchronization can be achieved by the ME latching, even with a multi-cycle insertion delay on the clock tree. By configuring Δ2 with a sufficient delay, the clock pause on *sClk* will rarely happen, and thus can be ignored. Consider a rising edge on the output signal of Δ1, denoted by Δ1+, in Figure 3.5. It leads *rClk* to be high. If *ri* is asserted simultaneously, the MUTEX might get into metastability. The worst situation occurs when *ri* wins the arbitration, which is indicated by *ai*+, after an unbounded resolution time. Until *ri* is de-asserted by the port controller, *gi* can turn to be high eventually. This results in a time interval between Δ1+ and *gi*+, as derived in (3.12). There $t_{ai=HIGH}$ is the on-phase duration of *ai*, and d_{MUTEX} the resolution time of the MUTEX, which includes both the intrinsic delay and the extra delay consumed for settling from metastability.

$$d_{\Delta 1+ \to gi+} = d_{AND} + d_{MUTEX} + t_{ai=HIGH} \,. \tag{3.12}$$

To prevent *sClk+* from being delayed, *gi* should be asserted no later than the output signal of Δ2 becomes high. This imposes a constraint on the delay of Δ2:

$$d_{\Delta 2} \geq d_{\Delta 1+->gi+}. \tag{3.13}$$

By optimizing the behavior of the asynchronous port controllers, the on-phase duration of *ai* can reach the minimally allowed pulse width for a latch, typically being less than five FO4 inverter delays. Taking (3.12) into (3.13) and ignoring d_{AND} and $d_{ai=HIGH}$, we obtain the lower bound of the delay on Δ2, which can be approximated based on the resolution time of MUTEX:

$$d_{\Delta 2} \approx d_{MUTEX}. \tag{3.14}$$

As addressed in Section 2.2.1, once getting metastable, the MUTEX will consume an indeterminate and unbounded time for settling. A practical solution is to estimate its resolution time according to the MTBF, as presented in (2.3). The condition of MTBF is application specific actually. Given a MTBF of 10,000 years for example, about forty FO4 inverter delays prove to be sufficient for a MUTEX exiting from metastability [46]. By configuring the delay on Δ2 to give an adequate time for resolving metastability, we can thus avoid the clock pause on *sClk*, yet achieving a negligible failure probability. It makes the timing constraint of (3.9) applicable also for the multi-cycle clock-tree delay. Therefore, taking (3.14) into (3.7), the MUTEX acknowledge window, i.e., the safe time region for the clock-tree insertion, can be obtained as:

$$w_{MUTEX} \approx T - d_{MUTEX}. \tag{3.15}$$

Given a certain clock period, the maximum resolution time can be reserved by bypassing Δ1 in the ring oscillator, as illustrated in Figure 3.11. Half a cycle is available for the MUTEX to settle without introducing clock pauses as a result. This indicates a practical constraint on the application of the ME latching synchronization in case of the multi-cycle clock-tree delay. That is, the time-safe synchronization cannot be ensured unless the resolution time, in accordance with the MTBF requirement, is less than half a clock period. The bypass of Δ1 leads to the minimum acknowledge window, relative to the clock period, as shown in Figure 3.12.

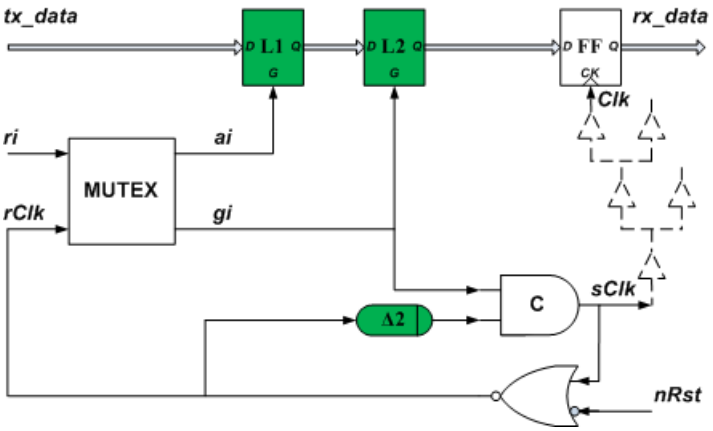


Figure 3.11 ME latching synchronization with Δ1 bypassed

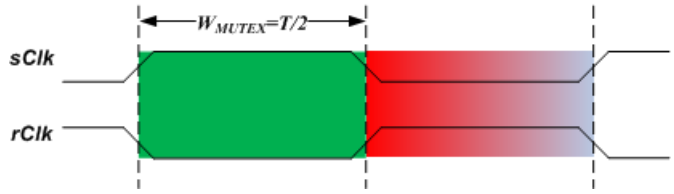


Figure 3.12 Minimum acknowledge window in ME latching synchronization

3.5 Summary

The following concludes our discussions in this chapter. First, the synchronization interface of ME latching proves to be reliable, under the constraint of (3.9). That is, the fractional clock-tree delay has to be smaller than the width of the acknowledge window. Second, in terms of sub-cycle clock-tree delays, value-safe, i.e., metastability-free, synchronization is guaranteed, and the acknowledge window can be optimized to be almost the entire clock cycle, as derived in (3.11). Third, given a multi-cycle clock-tree delay, time-safe synchronization, which is metastability negligible in other words, is allowed by configuring the acknowledge window of MUTEX in accordance with the resolution time of metastability, as expressed in (3.15).

In addition, it is noteworthy that, independent of the reliable synchronization issues, a deep clock tree with multi-cycle delays might be not in favor of GALS design. Mostly it is preferred to partition a complicated system into a number of clock domains, each working at an individual clock frequency. Often a set of local clock trees are therefore synthesized with rather short insertion delays. This highlights the practical importance of the value-safe synchronization, guaranteed by the ME latching, for the GALS design with sub-cycle clock-tree delays.

Chapter 4

Performance Analysis

In addition to reliability of synchronization, the performance requirements for asynchronous communication impose another design metric on the GALS interface circuits. The pausible clocking scheme has been criticized for years due to the potential limit of accommodating high-throughput data transfer. Clock-tree delays were found a serious performance bottleneck in [45] [47]. Actually, even ignoring the clock trees simulations on a typical GALS data link based on pausible clocking only lead to a throughput of at most one data item every two clock cycles [31]. A comparative analysis also indicated its relatively low throughput, among the representative GALS clocking strategies [48]. Up to now most performance studies of pausible clocking are performed by simulations, either at the circuit level [48] [49] or at the system level [50]. In contrast, an analytical model on the data cycle of boundary synchronizers was reported in [51].

This chapter presents a systematic analysis of the performance of GALS data links based on pausible clocking. The synchronization interface of ME latching is considered especially, due to its reliability. The input synchronization latency and the burst-mode data-link throughput are both analytically addressed. The critical design parameters that affect the performance, such as the clock-tree insertion delay, the I/O-port interconnect delay, the MUTEX acknowledge window and the TX vs. RX clock frequency ratio, are all taken into account. In particular, the upper bounds of the clock-tree and port-interconnect delays, which can be tolerated by the GALS data links without sacrificing the throughput, are derived. The analytical results are verified by simulation.

4.1 GALS Data Link Based on Pausible Clocking

A GALS data link, in this thesis, refers to a point-to-point communication channel between a pair of TX and RX blocks, which are timed by independent local clocks. A pausible clocking based GALS data link is abstracted in Figure 4.1. The shadowed units represent the functional core modules on both sides, which are traditional synchronous designs, each with an individual clock. We assume that each data item transferred over a data link has been latched by flipflops, both on output and on input. It addresses the standard design style of isolating the internal core logic from the external I/O delays in IP-based digital systems. The other units are all communication specific. They account for the overheads of a GALS data link, in the hardware as well as in the performance, when compared with the synchronous counterpart.

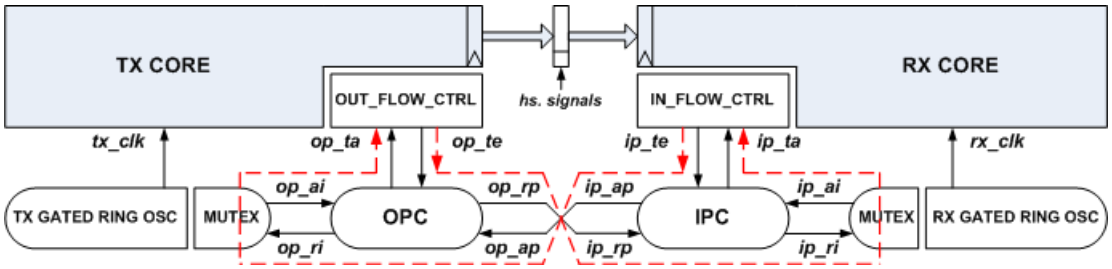


Figure 4.1 GALS data link based on pausible clocking

The red dashed lines in Figure 4.1 highlight a handshake loop, which schedules the data transfer from the TX to the RX. On each side there are three request-acknowledge channels surrounding an output (OPC) or input (IPC) port controller. Accordingly, each port controller can be characterized by three primary operations. First, it communicates with the core module through a flow control unit to arrange data flow over the data link. Any data item can be transferred only when the OPC and the IPC both get enabled by the flow control logic. Second, it times the local clock generator to synchronize its clock phases in accordance with the input data. For clarity, the MUTEX is explicitly depicted with the gated ring oscillator to represent the clock generator. Third, it performs handshakes with the partner port to execute each transfer of data item. The data bus from the TX to the RX is bundled with the handshake signals between the OPC and the IPC. Depending on the behavior of the port controllers, extra latches can be required on the data path to prevent data from being overwritten.

Above discussion indicates the central role of the port controllers in a GALS data link. Normally they account for the primary design consideration for high-performance communication. Plenty of work has been reported on the asynchronous port design for this reason. As addressed in Section 2.3.3, two families of ports can be characterized in general, named the demand-type ports and the poll-type ports [39]. Immediately upon being enabled, a demand-type port requests for a clock pause, regardless whether or not its partner port is activated for a data transfer. But a poll-type port will not suspend the clock, till detecting its partner port getting ready as well. Many early studies suggested implementing a GALS system using the demand-type ports to eliminate metastability in synchronization [32] [39] [45]. Actually, in two aspects the poll-type port based design deserves our attention. First, it alleviates the performance penalty for clock stretching. During the data transfer the clock is rarely gated by a poll-type port. Second, it imposes much relaxed timing constraints for practical applications, compared with the demand-type counterpart. In order to stop the clock as soon as getting enabled, a demand-type port only allows a sub-cycle delay for the clock tree insertion.

Thus, rather than making a comprehensive analysis, our work is focused on GALS data links equipped with the poll-type ports. We further assume that the ME latching in Figure 3.5 is applied on both the TX and the RX sides, considering the reliability issues. These summarize the main conditions of our performance analysis.

4.2 Input Synchronization Latency

For clarity, the timing parameters crucial for the ME latching synchronization are redrawn in Figure 4.2, where T is the clock period without the clock pause, w_{MUTEX} the MUTEX acknowledge window derived in (3.7), and Δ_{CT} the fractional clock-tree delay from the source clock *sClk* to the sampling clock *Clk* in (3.8). We will first address the synchronization latency suffered by the input data from being valid on the data bus to getting sampled by the functional modules.

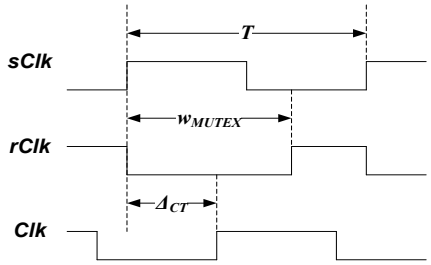


Figure 4.2 Timing parameters in ME latching synchronization

4.2.1 Synchronization Latency Function

Each time receiving an input data, the port controller generates a request of *ri*+ to the MUTEX for arbitration. Assuming that the *ri*+ appears $t \in [0, T)$ unit time after a *sClk*+, three scenarios can be distinguished on the data synchronization. First, consider a t meeting $0 \leq t < w_{MUTEX}$. That is, the *ri*+ arrives inside the MUTEX acknowledge window. It gets acknowledged immediately, and the data is sampled at the next *Clk*+. Thus little synchronization latency is introduced. Given that $w_{MUTEX} < t < T$, the *ri*+ is set outside the acknowledge window of the current cycle. It cannot be acknowledged until the MUTEX is released by the next *sClk*+. As a penalty, an additional cycle will be imposed on input data sampling. In the case of $t \approx w_{MUTEX}$, the MUTEX gets into metastability. As the *ri*+ has a 50% probability of winning, the synchronization latency is the average of the first two cases, as illustrated in Figure 4.3.

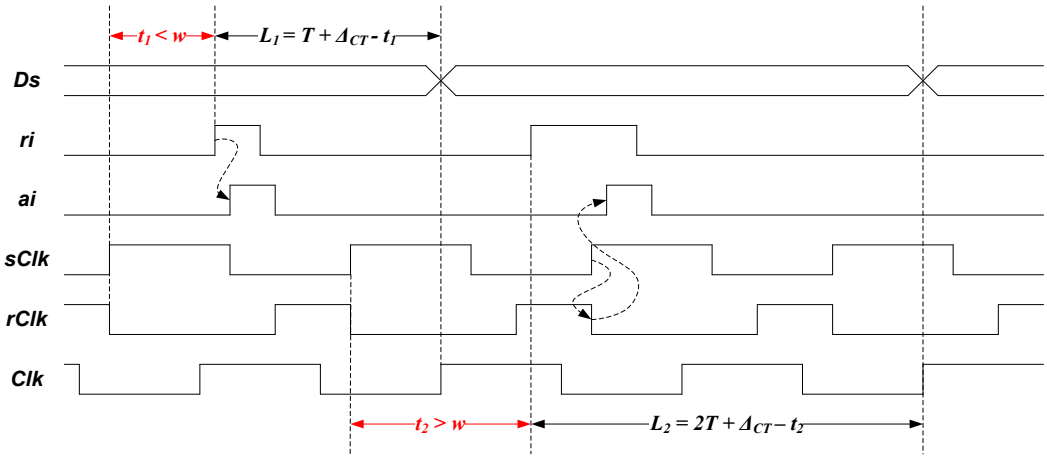


Figure 4.3 Synchronization latency for different input arrival time

Therefore, as presented in (4.1), the synchronization latency by ME latching can be expressed as a function of the input arrival time t, by taking the acknowledge window w_{MUTEX} and the clock-tree delay Δ_{CT} as design-specific parameters. In (4.1) T_W denotes the metastability window of a MUTEX. It is small, typically less than five FO4 inverter delays, and constant, mainly depending on the process [46].

$$L(t) = \begin{cases} T + \Delta_{CT} - t, \ t \in [0, w_{MUTEX} - \frac{T_W}{2}); \\ 3T/2 + \Delta_{CT} - w_{MUTEX}, \ t \in [w_{MUTEX} - \frac{T_W}{2}, w_{MUTEX} + \frac{T_W}{2}]; \\ 2T + \Delta_{CT} - t, \ t \in (w_{MUTEX} + \frac{T_W}{2}, T). \end{cases} \quad (4.1)$$

It can be seen that, given a Δ_{CT}, $L(t)$ is piecewise linear against t. The break-points of $L(t)$, however, are defined by w_{MUTEX}. The graphs of $L(t)$ regarding different Δ_{CT} and w_{MUTEX} are plotted in Figure 4.4. It is manifested that, to lower the latency of input synchronization, a wide MUTEX acknowledge window and a small fractional clock-tree delay are both beneficial.

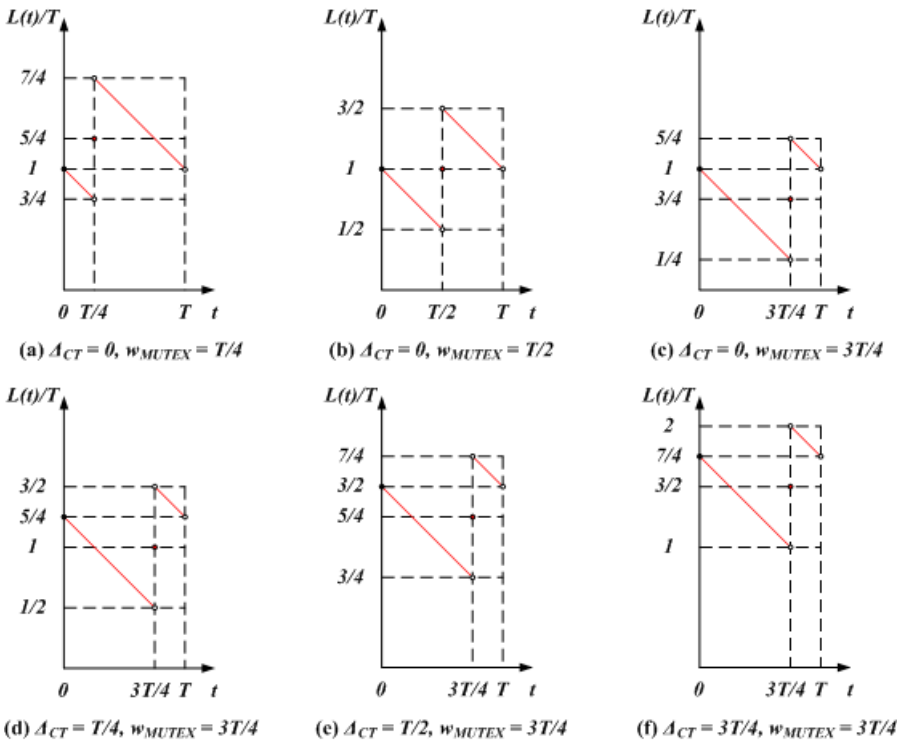


Figure 4.4 Graphs of $L(t)$ regarding different Δ_{CT} and w_{MUTEX}

It is of importance to notice that, for the reliable synchronization by ME latching, the timing constraint of (4.2) on w_{MUTEX} and Δ_{CT} has to be respected (see Figure 4.2). This presents a boundary condition for the design optimization.

$$\Delta_{CT} < w_{MUTEX} < T \ . \quad (4.2)$$

4.2.2 Synchronization Latency for Uniform Input Distribution

With the synchronization latency function $L(t)$, we can further address the average latency of synchronization for a large amount of data items transferred over a channel. Consider the simplest scenario for instance, where the input arrival time *t* is uniformly distributed over [0, *T*). The average latency can be derived as:

$$\overline{L} = \int_0^T \frac{1}{T} L(t)dt = (\frac{3}{2} - \frac{w_{MUTEX} - \Delta_{CT}}{T}) \cdot T \ . \tag{4.3}$$

We define $(w_{MUTEX} - \Delta_{CT})$ the effective acknowledge window. Then for an even distribution of input arrival time, the ME latching synchronization leads to an average latency monotonically decreasing with the effective acknowledge window. Taking into account (4.2), the effective acknowledge window is constrained by:

$$0 < w_{MUTEX} - \Delta_{CT} < T \ . \tag{4.4}$$

Thus, substituting (4.4) to (4.3), the lower and upper bounds of the average latency, given a uniform distribution of input over time, can be obtained as:

$$\frac{1}{2}T < \overline{L} < \frac{3}{2}T \ . \tag{4.5}$$

In particular, given that $(w_{MUTEX} - \Delta_{CT}) > T/2$, the ME latching synchronization will achieve a sub-cycle latency on average. This accounts for a significant advantage over the traditional GALS interface circuits. Taking the double-flipflop synchronizer as an example, an average synchronization latency of $\overline{L} = 3T/2$ will be introduced, which actually addresses the worst case of the ME latching.

4.3 Average Data Throughput

4.3.1 Handshake Loop delay

Besides the input synchronization latency, the data throughput contributes another performance metric to be considered by the GALS data link design. Assuming that the TX core and the RX core both allow burst-mode communication, i.e., being capable of sending and receiving a data item in each clock cycle, we will investigate the throughput affordable for the data link of Figure 4.1.

Our discussion is restricted to the case that the two-phase bundled-data protocol is employed by the handshaking between the I/O port controllers (*op/ip_rp* and *op/ip_ap*). It addresses the typical design solution when data throughput is the concern. Under this condition, the cycle time required for transferring a data item over the data link can be effectively measured by the time interval between two consecutive transitions on *op_rp*. As illustrated in Figure 4.1, it is actually determined by the delay of the handshake loop, named T_{Loop}, between the TX and the RX. In principle, T_{Loop} consists of three distinct delay components, as formulated in (4.6):

$$
\begin{aligned}
T_{Loop} &= d_{IPC} + d_{OPC} + d_{INT};\\
d_{IPC} &= d_{ip\_rp->ip\_ap};\\
d_{OPC} &= d_{op\_ap->op\_ip};\\
d_{INT} &= d_{op\_rp->ip\_rp} + d_{ip\_ap->op\_ap}.
\end{aligned}
\tag{4.6}
$$

In (4.6), d_{IPC} represents the response delay of the IPC from receiving a request on *ip_rp* to sending an acknowledge signal on *ip_ap*. Similarly, d_{OPC} denotes the response delay of the OPC from getting acknowledged on *op_ap* to releasing the next request on *op_rp*. In addition, d_{INT} accounts for the round-trip interconnect delay between the I/O port controllers, which is fixed after the layout.

4.3.2 Data-Link Throughput

It is noteworthy that T_{Loop} can vary from one data cycle to another. A main reason lies in the different phase shifts between the TX and the RX clocks [51]. Rather than deriving the exact T_{Loop} for each data cycle, we are interested in the average loop delay, represented by $\overline{T_{Loop}}$, during continuously transferring a large amount of data. It can be obtained by (4.7), assuming that the I/O ports react independently of each other, with the average response delays of $\overline{d_{IPC}}$ and $\overline{d_{OPC}}$, respectively:

$$
\overline{T_{Loop}} = \overline{d_{IPC}} + \overline{d_{OPC}} + d_{INT}\,. \tag{4.7}
$$

Because any data transfer can be performed only when it gets enabled by the flow control units on both sides, there simply exists a lower bound of the average loop delay:

$$
\overline{T_{Loop}} \geq \max(T_{TX}, T_{RX})\,. \tag{4.8}
$$

If $\overline{T_{Loop}}$ reaches its lower bound, the data-link throughput is thus determined by the processing speeds of the TX and the RX functional modules. As a result, no throughput loss is caused due to the asynchronous communication. This represents the condition of optimal GALS data-link design from the throughput point of view. Once $\overline{T_{Loop}}$ is larger than the clock periods, the throughput will be slowed down due to the low handshaking speed between the TX and the RX. This gives rise to the performance loss.

Given a certain $\overline{T_{Loop}}$, the throughput of a data link, when measured on the RX side, can be specified by (4.9). It indicates the average number of data items received by the RX core in each clock cycle:

$$
\theta_{DL} = \frac{T_{RX}}{\overline{T_{Loop}}}\,. \tag{4.9}
$$

Obviously, according to (4.8), the data-link throughput is no more than one:

$$
\theta_{DL} \leq 1\,. \tag{4.10}
$$

4.4 Throughput Modeling of GALS Data Links

4.4.1 Tightly-Coupled Data Link

Take a particular data link for instance. The STG specifications of its OPC and IPC are shown in Figure 4.5. We name it tightly-coupled data link, considering the fact that, as indicated by the red dashed lines, each transition of the I/O port controllers is in fact executed in sequence. Early work studied its performance by simulations [31] [48]. In this section, we would like to derive an analytical model of its throughput, based on the average loop delay of handshaking.

The operations of the IPC can be briefed as follows. Once receiving a port request (*ip_rp*+/-), it raises an internal request (*ip_ri*+) to the MUTEX for a clock pause. This starts the input synchronization on the RX side. After getting acknowledged (*ip_ai*+), the IPC produces a transfer acknowledge (*ip_ta*+/-) to signify the valid input data. Then the internal request can be de-asserted (*ip_ri*-), thus releasing the RX clock (*ip_ai*-). On the next clock rising edge, the IPC gets activated by the RX flow control unit (*ip_te*+/-) to acknowledge the OPC (*ip_ap*+/-). In the mean time, the input data is sampled by the RX core module.

As a result, the delay from *ip_ri*+ to *ip_te*+/- addresses the input synchronization latency on the RX side. Given that the ME latching synchronization is applied, it can be obtained by (4.11), where $L_{RX}(t)$ denotes the synchronization latency function of (4.1), in terms of the RX design specific parameters:

$$d_{ip\_ri+->ip\_te} = L_{RX}(t). \tag{4.11}$$

Taking (4.11) into (4.6), we can further derive the response delay of the IPC as:

$$\begin{aligned} d_{IPC} &= L_{RX}(t) + \varepsilon_{RX} \\ \varepsilon_{RX} &= d_{ip\_rp->ip\_ri+} + d_{ip\_te->ip\_ap} \end{aligned}. \tag{4.12}$$

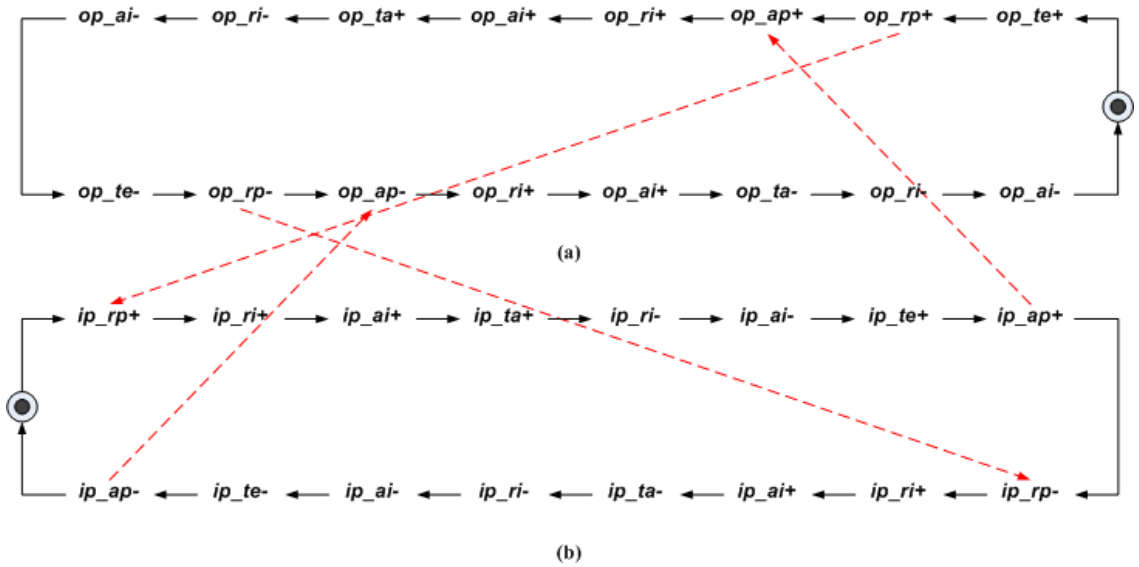


Figure 4.5 STG of the tightly-coupled data link: (a) OPC and (b) IPC

The operations of the OPC are exactly the same as the IPC. In particular, here it is the acknowledge feedback (*op_ap+/-*) from the IPC causing the input synchronization. A transfer acknowledge (*op_ta+/-*) is released as a result, which identifies the end of a transfer over the data link. Similar to (4.12), the response delay of the OPC thus can be expressed based on the synchronization latency function $L_{TX}(t)$:

$$\begin{aligned} d_{OPC} &= L_{TX}(t) + \varepsilon_{TX} \\ \varepsilon_{TX} &= d_{op\_ap->op\_ri+} + d_{op\_te->op\_rp} \end{aligned} . \tag{4.13}$$

The variables t in (4.12) and (4.13) measure the arrival time of *ip_ri+* and *op_ri+* relative to the rising edges of the RX and the TX clocks, respectively. Provided that the TX core and the RX core are timed asynchronously, t follows the uniform distribution in each of clock cycles. According to (4.3), we obtain the average input synchronization latency of the I/O port controllers as follows:

$$\begin{aligned} \overline{L_{RX}} &= (\frac{3}{2} - \frac{w_{MUTEX}^{RX} - \Delta_{CT}^{RX}}{T_{RX}}) \cdot T_{RX} \\ \overline{L_{TX}} &= (\frac{3}{2} - \frac{w_{MUTEX}^{TX} - \Delta_{CT}^{TX}}{T_{TX}}) \cdot T_{TX} \end{aligned} . \tag{4.14}$$

In comparison with the average synchronization latency, the internal propagation delays of the port controllers, i.e., ε_{RX} in (4.12) and ε_{TX} in (4.13), are usually negligible. It results from the concise structures of the IPC and the OPC after synthesis, as will be addressed in Section 4.4.3. For this reason, (4.15) presents a reasonable approximation of the average response delay of each port controller:

$$\begin{aligned} \overline{d_{IPC}} &\approx \overline{L_{RX}} \\ \overline{d_{OPC}} &\approx \overline{L_{TX}} \end{aligned} . \tag{4.15}$$

Taking (4.14) and (4.15) into (4.7), an analytical model of the average handshake loop delay of the tightly-coupled data link is achieved by (4.16). All the performance-critical parameters have been taken into account, including the MUTEX acknowledge window, the clock-tree insertion delay, the I/O-port interconnect delay and the TX/RX clock frequencies.

$$\overline{T_{Loop}} \approx (\frac{3}{2} - \frac{w_{MUTEX}^{TX} - \Delta_{CT}^{TX}}{T_{TX}}) \cdot T_{TX} + (\frac{3}{2} - \frac{w_{MUTEX}^{RX} - \Delta_{CT}^{RX}}{T_{RX}}) \cdot T_{RX} + d_{INT} . \tag{4.16}$$

Furthermore, substituting (4.16) into (4.9), the throughput allowed by the tightly-coupled data link can be expressed in terms of various design-specific parameters. On the other side, however, we are more interested in the timing condition to avoid throughput loss. Substituting (4.16) into (4.8), the following constraint can be thus derived on the effective acknowledge window:

$$(w_{MUTEX}^{TX} - \Delta_{CT}^{TX}) + (w_{MUTEX}^{RX} - \Delta_{CT}^{RX}) > \frac{1}{2}\max(T_{TX}, T_{RX}) + \frac{3}{2}\min(T_{TX}, T_{RX}) + d_{INT} . \tag{4.17}$$

4.4.2 Loosely-Coupled Data Link

Rather than in the tightly-coupled data link, the activity of the I/O port controllers can be efficiently decoupled by introducing concurrency. Figure 4.6 illustrates the STG specification of a loosely-coupled data link for example. The handshaking sequence in the IPC is rescheduled comparing with Figure 4.5. It is the transition on *ip_ta*, instead of on *ip_te*, that triggers *ip_ap* to toggle. That means, the IPC will send an acknowledge feedback to the OPC immediately when a data item is admitted to be sampled, instead of it getting sampled. The synchronization on the RX side (from *ip_ri*+ to *ip_te*+/-) is therefore carried out partially in parallel with the synchronization on the TX side (from *op_ri*+ to *op_te*+/-). This accounts for the loose coupling of the data link. Rather than deriving a throughput model, below we are going to focus on the timing conditions for preventing throughput loss in burst mode.

A. TX-Limited Data Link

First, consider the scenario of $T_{TX} > T_{RX}$. In this case the maximum throughput of the data link is limited by T_{TX} according to (4.8). To transmit one datum per cycle, the OPC has to get acknowledged (*op_ap*+/-) from the handshake loop within the effective acknowledge window of each TX clock cycle. This leads to the following constraint on the maximum response delay of the IPC:

$$\max(d_{IPC}) < (w_{MUTEX}^{TX} - \Delta_{CT}^{TX}) - \varepsilon_{TX} - d_{INT} . \tag{4.18}$$

Assume that *ip_rp*+/- arrives simultaneously with *rClk*+ as shown in Figure 4.7(a). It cannot get acknowledged until *rClk* becomes low. And the following *ip_rp*-/+, which comes in a TX clock cycle, has to wait to be responded until *ip_te*-/+ is triggered by the next RX clock rising edge. This incurs the maximum response delay of the IPC:

$$\max(d_{IPC}) = (T_{RX} + \Delta_{CT}^{RX}) - (T_{TX} - (T_{RX} - w_{MUTEX}^{RX})) + \varepsilon_{RX} . \tag{4.19}$$

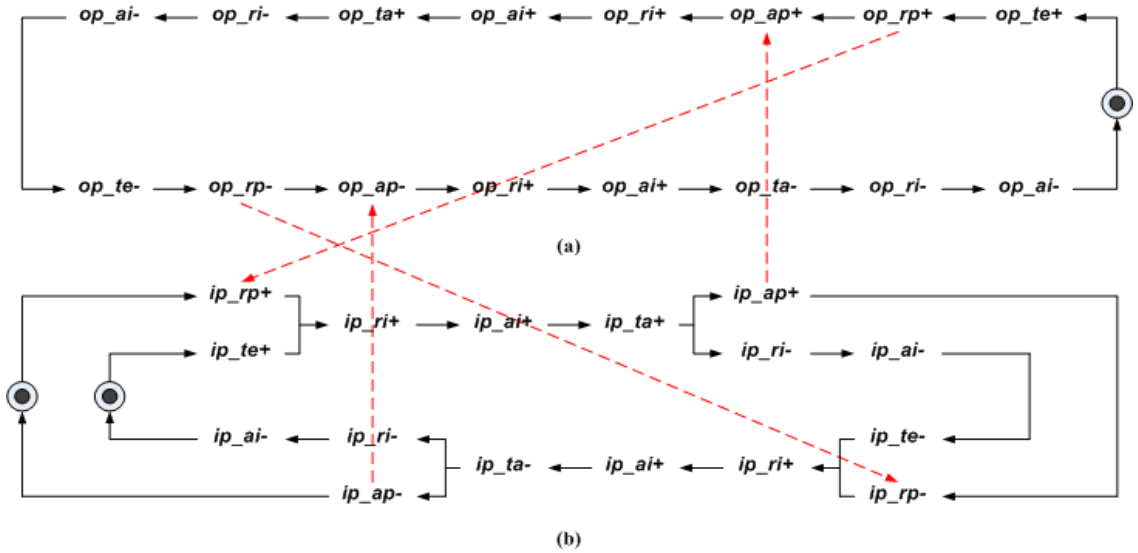


Figure 4.6 STG of the Loosely-coupled data link: (a) OPC and (b) IPC

Substituting (4.19) into (4.18) and ignoring ε_{RX} and ε_{TX}, we can obtain the timing condition of (4.20) on the effective acknowledge windows, under which no throughput drop is introduced due to the handshake loop delay, given that $T_{TX} > T_{RX}$:

$$(w_{MUTEX}^{TX} - \Delta_{CT}^{TX}) + (w_{MUTEX}^{RX} - \Delta_{CT}^{RX}) > (2T_{RX} - T_{TX}) + d_{INT} . \quad (4.20)$$

B. RX-Limited Data Link

Similar analysis can be performed in the case of $T_{RX} > T_{TX}$, where the data-link throughput is limited by T_{RX}. We expect to reach $\theta_{DL} = 1$, i.e., receiving one data item every cycle. For this reason, the IPC has to get a request (*ip_rp+/-*) within the effective acknowledge window of each RX clock cycle. Once an *ip_rp+/-* arrives before the IPC being enabled, the corresponding *ip_ap+/-* needs to wait until the next RX clock rising edge to be triggered by the *ip_te+/-*. To guarantee that the following *ip_rp-/+* will get acknowledged, the maximum response delay of the OPC should satisfy:

$$\max(d_{OPC}) < T_{RX} + w_{MUTEX}^{RX} - \Delta_{CT}^{RX} - \varepsilon_{RX} - d_{INT} . \quad (4.21)$$

The OPC incurs the maximum response delay if an *op_ap+/-* appears at the same time with *rClk+* (see Figure 4.7(b)). It leads to an *op_rp+/-* in another TX clock cycle:

$$\max(d_{OPC}) = 2T_{TX} + \Delta_{CT}^{TX} - w_{MUTEX}^{TX} + \varepsilon_{TX} . \quad (4.22)$$

According to (4.21) and (4.22), given that $T_{RX} > T_{TX}$, the below timing condition to avoid throughput loss can be derived for the loosely-coupled data link:

$$(w_{MUTEX}^{TX} - \Delta_{CT}^{TX}) + (w_{MUTEX}^{RX} - \Delta_{CT}^{RX}) > (2T_{TX} - T_{RX}) + d_{INT} . \quad (4.23)$$

C. Generalized Timing Condition

Taking (4.20) and (4.23) both into account, a generalized condition for throughput-tolerant data transfer by the loosely-coupled data link can be derived:

$$(w_{MUTEX}^{TX} - \Delta_{CT}^{TX}) + (w_{MUTEX}^{RX} - \Delta_{CT}^{RX}) > (2\min(T_{TX}, T_{RX}) - \max(T_{TX}, T_{RX})) + d_{INT} . \quad (4.24)$$

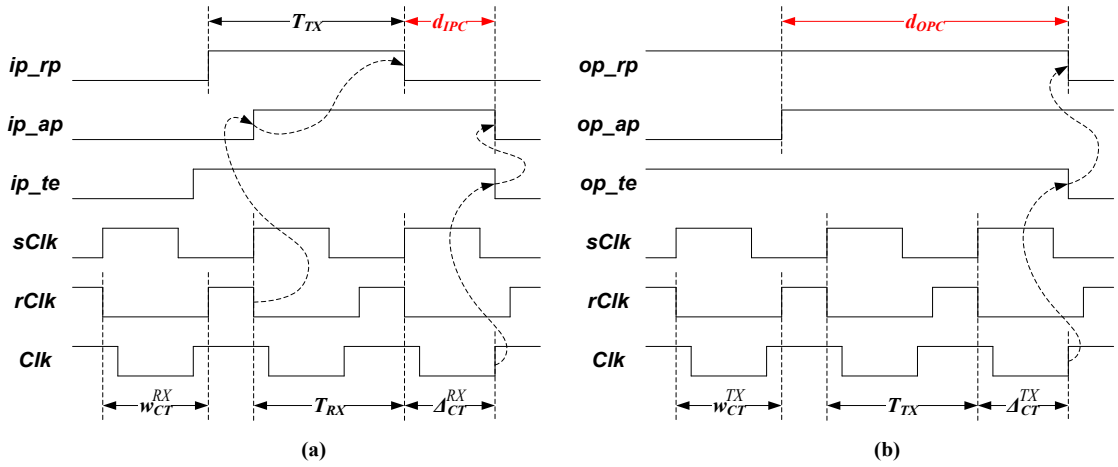


Figure 4.7 Maximum response delays of (a) the IPC and (b) the OPC

4.4.3 Comparisons

Based on above analysis, we would like to have a comparison in terms of throughput between the tightly-coupled and the loosely-coupled data links. Attention has been paid in particular on the scenario of plesiochronous clocking, i.e.:

$$T_{TX} \approx T_{RX} \approx T . \tag{4.25}$$

Numerous simulations indicate that in this case the data-link throughput is susceptible seriously to the handshake loop delay [31] [48]. In other words, it addresses the worst condition, regarding the throughput, of the GALS data-link design.

According to (4.25), (4.17) for the tightly-coupled data link can be simplified as:

$$(w_{MUTEX}^{TX} - \Delta_{CT}^{TX}) + (w_{MUTEX}^{RX} - \Delta_{CT}^{RX}) > 2T + d_{INT} . \tag{4.26}$$

In comparison, (4.24) for the loosely-coupled data link turns out to be:

$$(w_{MUTEX}^{TX} - \Delta_{CT}^{TX}) + (w_{MUTEX}^{RX} - \Delta_{CT}^{RX}) > T + d_{INT} . \tag{4.27}$$

Essentially, (4.26) and (4.27) present the upper bounds of the clock-tree and interconnect delays for the plesiochronous data communication. In quite concise forms, they point out the directions to optimize the design and implementation of GALS data links. The loosely-coupled data link outperforms the tightly-coupled one, by allowing a much more relaxed timing constraint, both for the clock-tree synthesis and for the port interconnect routing. Given a negligible interconnect delay for example, it almost halves the throughput-tolerant condition of the effective acknowledge windows.

On the other side, however, any optimization on the data link has to pay respects to the boundary condition of (4.2). It is imposed for guaranteeing reliable synchronization, as aforementioned. Taking (4.2) into account, we see that (4.26) never can be achieved, regardless of the clock-tree insertion and port interconnect delays. That is, even with a slight frequency mismatch on the TX and RX clocks, the tightly-coupled data link will suffer from a remarkable throughput drop. As a comparison, by reserving sufficient acknowledge windows, the loosely-coupled data link design turns out to be tolerant of the clock-tree and interconnect delays, to certain degree. The concurrent operations of the port controllers thus make possible the high-throughput communication in practice.

For a reasonable comparison, it is necessary to also address the design complexity of the I/O port controllers in the tightly-coupled and the loosely-coupled data links. We can apply *Petrify* to synthesize the port controllers based on the STG specifications [38]. The following figures illustrate the gate-level circuits derived by *Petrify*, in accordance with the IHP 130-nm CMOS standard cells for example. Each circuit presented there is actually an asynchronous FSM. Its state is held by the combinational loops, leading to a quite simple structure. This accounts for the negligible propagation delays, ε_{TX} and ε_{RX}, of the I/O port controllers as aforementioned. More important, little hardware overhead is introduced by the loosely-coupled design of the I/O port controllers.

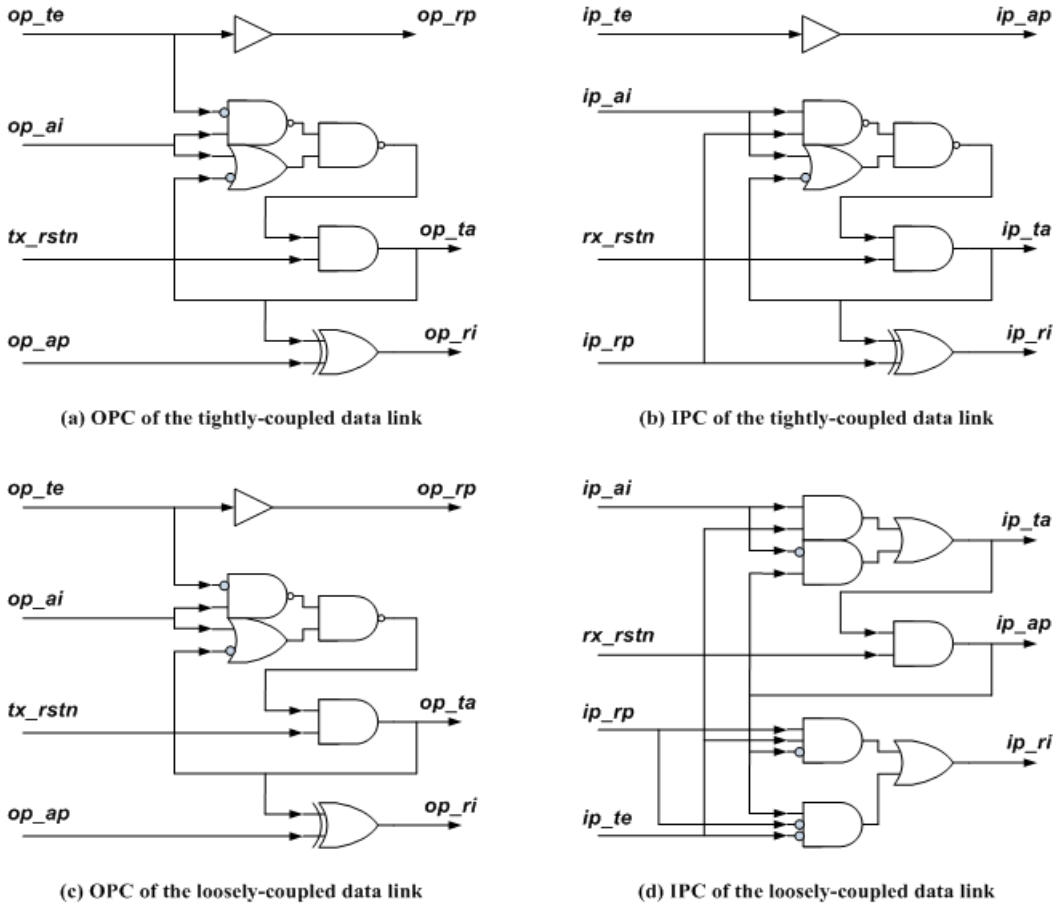


Figure 4.8 Gate-level designs of the asynchronous I/O port controllers:

4.5 Performance Evaluation of GALS Data Links

Intensive simulations have been performed to evaluate the accuracy of our analysis of the GALS data-link throughput. Both, the TX core and the RX core, are designed to support the burst-mode data transfer, i.e. being capable to send and receive a data item per cycle. The post-synthesis circuits of the I/O port controllers, as illustrated in Figure 4.8, were employed to design the tightly- and loosely-coupled data links in simulations. Each time a total of $64k$ items of data were continuously transferred via a data link. The number of clock cycles *N* needed by the RX in each burst was measured. The data-link throughput thus can be calculated by $64k/N$. Under different clock-tree and port inter-connect delays, the analytical estimations were verified by comparing with the simulation results. For a clear comparison, the internal propagation delays, ε_{TX} and ε_{RX}, of the port controllers were neglected in the experiments. We will elaborate on the practical circuit designs of the GALS data links based on pausible clocking in Chapter 5.

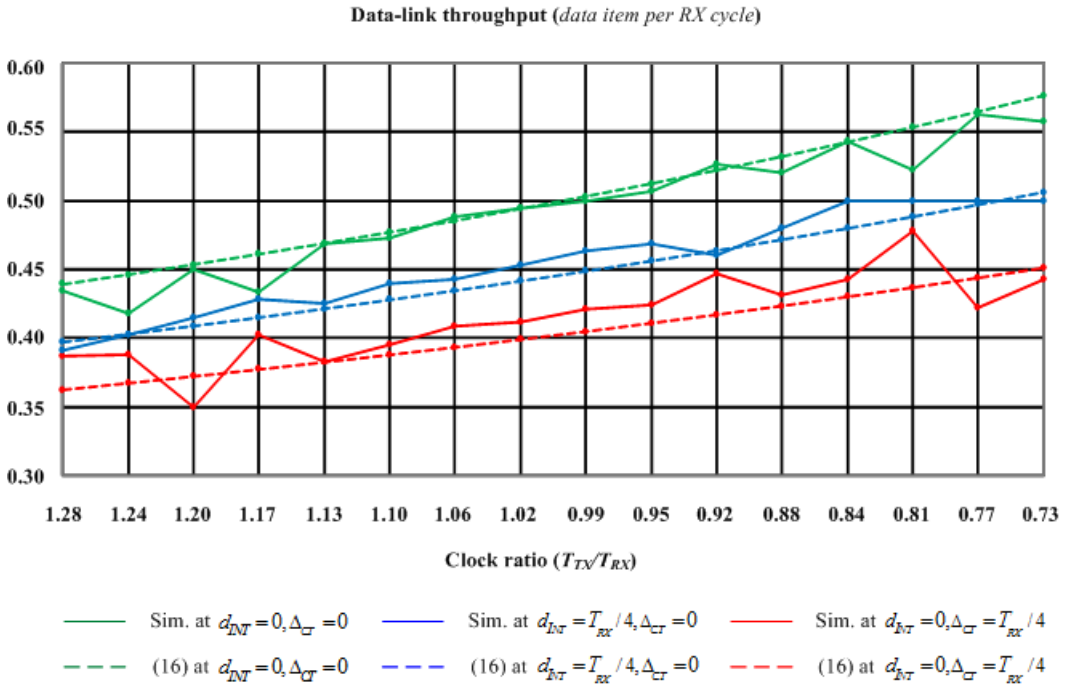


Figure 4.9 Throughput of the tightly-coupled data link

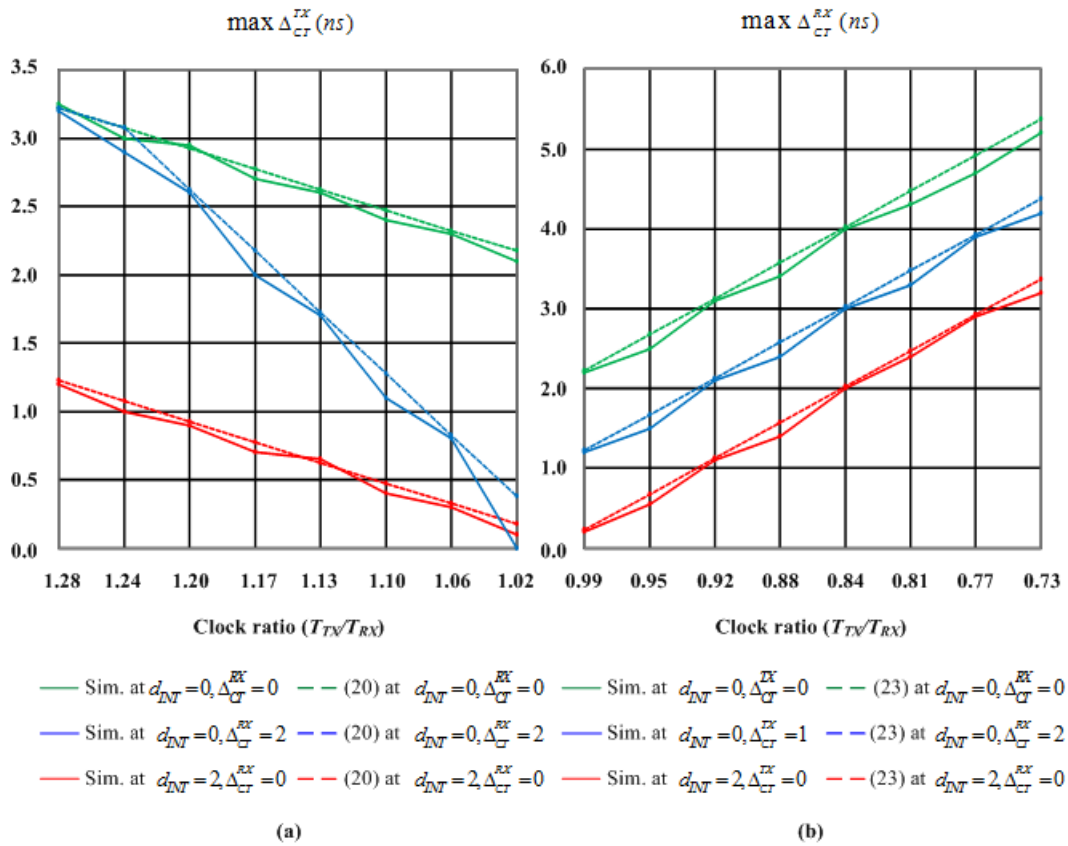


Figure 4.10 Maximum clock-tree delays tolerated by the loosely-coupled data link at (a) $T_{TX} > T_{RX}$ and (b) $T_{TX} < T_{RX}$

The tightly-coupled GALS data link was first evaluated by simulation. The RX clock was fixed at $T_{RX} = 8.33\ ns$, i.e., *120 MHz*, and the TX clock was adjusted with a clock ratio of $0.7 < T_{TX}/T_{RX} < 1.3$. Plesiochronous clocking was thus addressed in particular. The MUTEX acknowledge windows were set to be $w_{MUTEX}^{RX} = 0.75T_{RX}$ and $w_{MUTEX}^{TX} = 0.5T_{TX}$. Figure 4.9 illustrates the comparison between the simulation and the analytical model of (16) in terms of data-link throughput. As can be seen, our model matched the simulation results pretty well, with an error of 8.5% at most, at different clock ratios and clock-tree/interconnect delays. The experiments also manifest the poor performance of the tightly-coupled data link. Even if the TX and the RX run at virtually the same speed ($T_{TX}/T_{RX} = 0.99$ in Figure 4.9), the channel only allowed a throughput of at most 0.5 data item per cycle. As revealed in (16), this is caused by the large handshake loop delay due to the synchronization latency on both sides.

The throughput-tolerant clock-tree delay was validated for the loosely-coupled data link, as presented in Figure 4.10. The maximum clock-tree delay, accommodated by the data link, is found highly dependent on the clock ratio of T_{TX}/T_{RX}. In the condition of plesiochronous clocking, i.e. $T_{TX}/T_{RX} \approx 1$, only a tiny clock-tree delay can be tolerated. However, with the increasing discrepancy between T_{TX} and T_{RX}, it turns out possible to introduce a considerable delay on the clock tree without sacrificing data-link throughput. Again, plesiochronous clocking is shown to address the most challenging condition for the high-throughput GALS data-link design.

4.6 Summary

For the first time, a quantitative analysis is presented on the performance of GALS data links based on pausible clocking. All performance-critical parameters are taken into account, including the clock-tree insertion delay, the I/O-port inter-connect delay, the MUTEX acknowledge window and the TX vs. RX clock frequency ratio. By alleviating the penalty of synchronization latency, the ME latching proves to be beneficial for the GALS data-link design. Given input data with a uniformly distributed arrival time, the sub-cycle synchronization latency is achievable. Furthermore, the loosely-coupled data-link design is proposed. It introduces concurrency on the input synchronization between the TX and the RX. This significantly contributes to avoid throughput loss due to handshake loop delay. Our work paves the way to the optimal design and application of the pausible clocking scheme.

The derivation of the synchronization latency function in (4.1) ignored the metastability effects of the MUTEX. Metastability occurs when the input data occasionally arrives in the MUTEX metastability window T_W. It might require the clock pause to be resolved, leading to extra synchronization latency. However, since T_W is typically tiny in comparison with the clock period, the clock pause occurs rarely. It is our opinion that, metastability in fact has little impact on the performance of GALS interface circuits.

It is noteworthy that full decoupling of the data link can be obtained by inserting an asynchronous FIFO between the I/O port controllers. This is of particular importance if the I/O-port interconnect delay is comparable to, or even larger than, the clock periods. In this case, the handshake loop delay in (4.7) is dominated by the interconnect delay. Then reducing the response delays of the I/O ports, as addressed by the loosely-coupled data-link design, would barely improve throughput. With a linear asynchronous FIFO, the handshake loop between the I/O port controllers can be split into a number of sub-loops, which work concurrently to each other. The *Mousetrap* FIFO design actually fits our requirements very well. It allows the bundled-data communication with two-phase handshaking between the I/O ports [52]. However, overheads in power and silicon area can be introduced. A latest overview on the design of high-performance asynchronous pipelines has been presented in [53].

Chapter 5

Circuit Design of GALS Data Links

This chapter elaborates on the implementation issues of the GALS data links based on pausible clocking. The reliability, performance and hardware overheads account for the major design metrics to be respected. To address different applications, two typical solutions are proposed: high throughput design of the loosely-coupled data link and low overhead design of the tightly-coupled data link. Also presented is the timing analysis on each of GALS data-link designs.

Although a variety of circuit designs for pausible clocking based GALS data links were reported in the literature, they are mostly optimized for specific design conditions. The design of flow control logic introduces a critical challenge, especially for the high-throughput communication, which was often overlooked by the previous work. To support the burst-mode data transfer, many studies suggested the demand-type port designs. Under this circumstance, flow control can be implicitly done by clock gating, leading to a full saving of the control unit. As discussed in Section 4.1, this design style, however, imposes a timing constraint of sub-cycle clock-tree delays on the synchronous modules. Our work distinguishes itself by exploring the more efficient and flexible flow control based on the poll-type I/O ports. No assumption is inferred on the clock-tree insertion delays. This actually denotes the general significance of our work for the GALS design by pausible clocking.

5.1 High-Throughput Design of Loosely-Coupled Data Link

Figure 5.1 illustrates an implementation of the loosely-coupled data-link design for high-throughput data transfer. The pausible clock generator and the I/O port controllers are presented by the shadowed boxes. The gate-level circuits can be found in Figure 3.5 and Figure 4.8, respectively. As a contrast, the flow control logic, both on the TX side and on the RX side, is drawn in detail. The flipflops in each control unit are triggered by the individual local clock. Due to the lack of space, we omitted the function modules in the figure, and only the TX output register FF_{TX} and the RX input register FF_{RX} are depicted explicitly (highlighted in red). Three stages of intermediate latches are inserted between FF_{TX} and FF_{RX}, which are controlled by the asynchronous handshake signals between the port controllers. Below we are going to further address the implementation details of the loosely-coupled data link, with a particular attention on the efficient and flexible control of data flow.

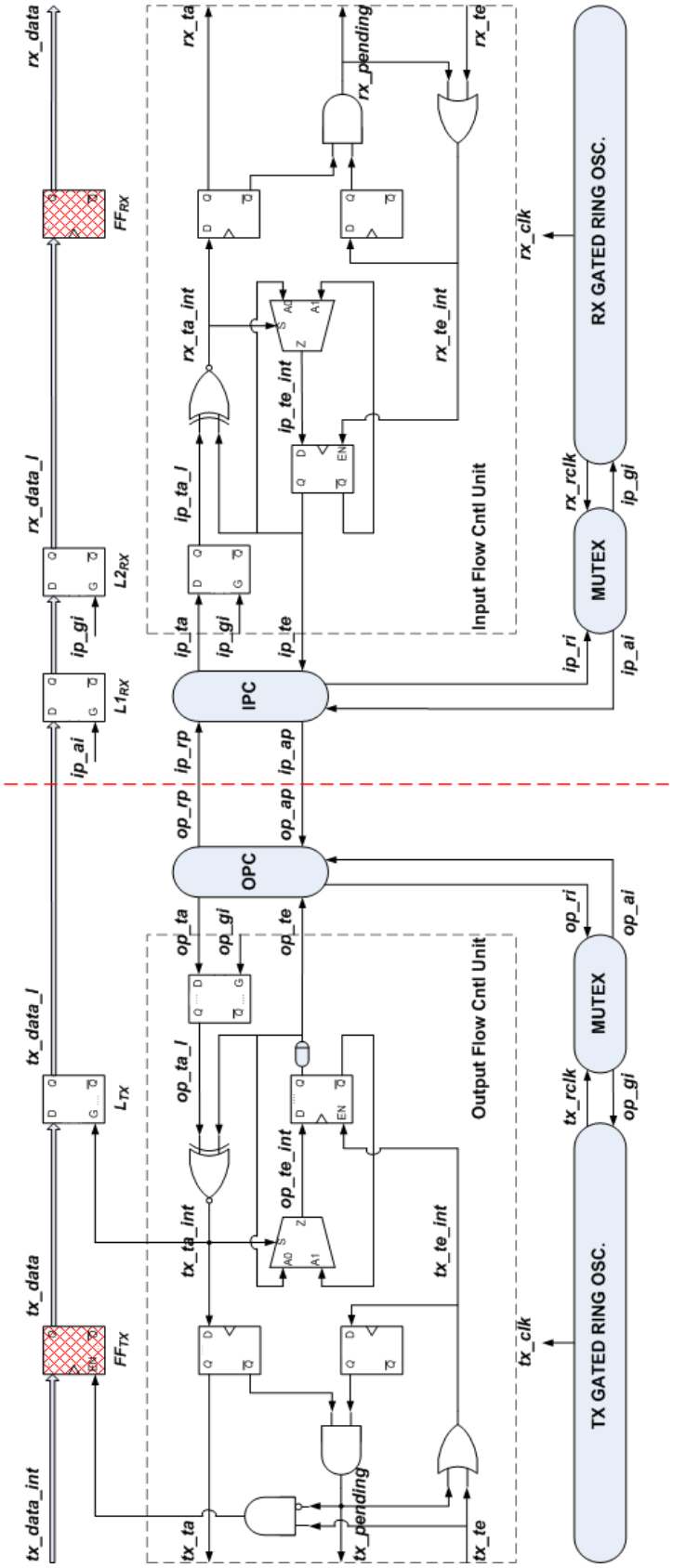


Figure 5.1 High-throughput design of the loosely-coupled GALS data link

5.1.1 Synchronous Core Modules

The TX and RX core modules can be abstracted as shown in Figure 5.2. Take the TX design for example. The shadowed blocks represent the TX FSM, which is timed by the TX local clock *tx_clk*. Its logic design is determined by the TX functionality, independent of the GALS design. The communication between the TX FSM and the output flow control unit is performed by a pair of handshake signals, *tx_te* and *tx_ta*, using the four-phase protocol. Once an item of data, *tx_data_int*, is ready to be transmitted, *tx_te* is asserted by the FSM to enable the flow control unit for a data transfer. Until the transfer is done, *tx_ta* is set to be high by the flow control unit to acknowledge the FSM. Another control signal, *tx_pending*, is also produced by the output flow control logic. It becomes high in case that the data link is currently occupied by a data transfer. At this moment, an outgoing request of data transfer, which is denoted by *tx_te = 1*, will lead the TX FSM to be stopped. We can hence avoid losing data. In contrast, as long as *tx_te = 0*, meaning that no additional transfer has to be carried out, the FSM can continue its operation without getting interrupted. This alleviates the impact of pending transfers on the TX functional processing. Similar analysis also can be applied on the design of RX core modules.

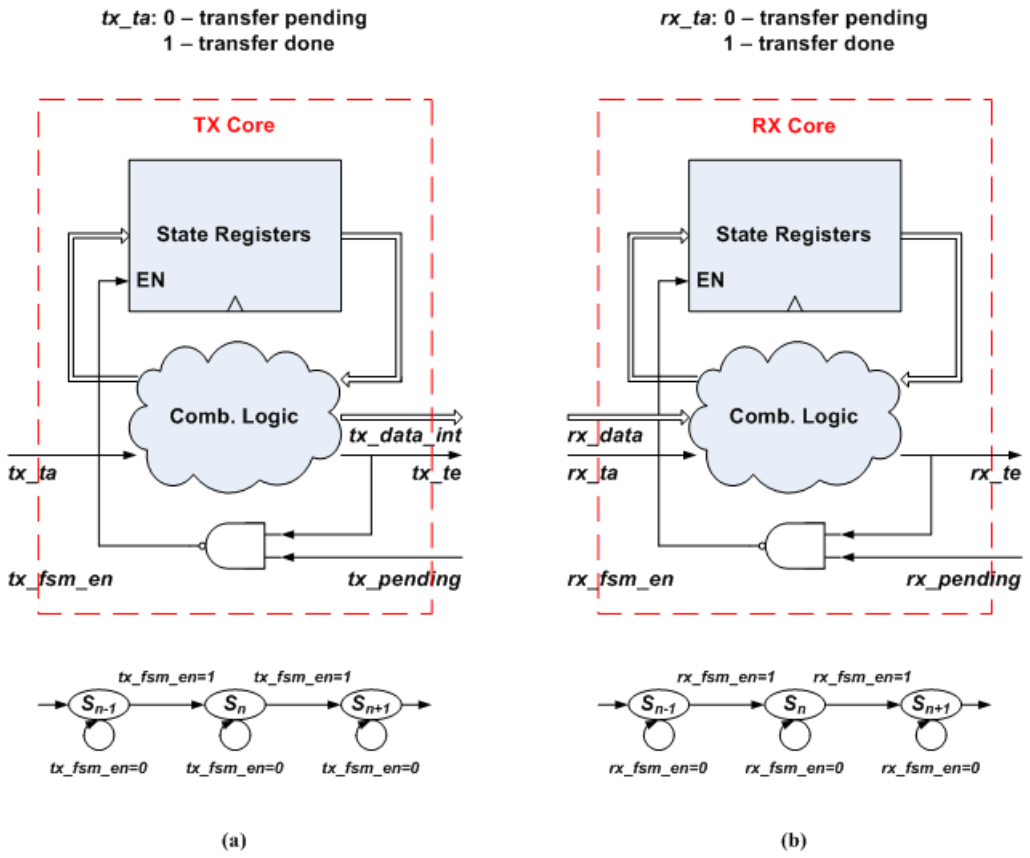


Figure 5.2 Design abstracts of the TX and the RX core modules

5.1.2 Flow Control Units

The flow control circuits have been illustrated in Figure 5.1. It is worth to note that our design is fully based on standard logic gates. Each control unit is equipped with two sets of handshake interfaces: *tx/rx_te* and *tx/rx_ta* with the TX/RX FSMs as mentioned above, and *op/ip_te* and *op/ip_ta* with the I/O port controllers. Especially, to facilitate burst-mode data transfer, the two-phase handshake protocol is employed on the signals *op/ip_te* and *op/ip_ta*.

Consider the output flow control on the TX side first. Its design can be much more complex than expected. An fundamental challenge is to allow the TX FSM to send data continuously. Actually, as pointed out in [31], the TX FSM has to wait one clock cycle at least from raising a *tx_te* to sampling the corresponding *tx_ta*. This would limit the TX data rate to be at maximum one data every two cycles. An asynchronous FIFO can be inserted to improve the throughput, yet at a cost of hardware overhead. Alternatively, we present a concise design scheme to address this issue. As depicted in Figure 5.1, the TX output data, *tx_data*, is buffered by an additional stage of latch L_{TX}. The gate signal of L_{TX}, i.e., *tx_ta_int*, is derived by a simple XNOR operation between the transfer enable *op_te* and the latched transfer acknowledge *op_ta_l*. L_{TX} is locked (*tx_ta_int* = 0) as soon as a transfer gets enabled (*op_te*+/-), and it is unlocked (*tx_ta_int* = 1) after the transfer gets acknowledged (*op_ta*+/-). Therefore, a stable output from L_{TX} during each data transfer can be guaranteed. Rather than waiting one cycle for detecting the status of *tx_ta*, the TX FSM moves into the next state immediately when an item of data gets latched by L_{TX}. Given that the latched data is transferred within one TX clock cycle, the FSM can always launch a new transfer in the subsequent cycle. Continuous data sending is allowed as a result. In contrast, a pending transfer will lead *tx_ta* to be low in the next clock cycle. Simultaneously, *tx_pending* will be raised to hold the TX FSM if necessary. Under this condition, the associated output data is stored in FF_{TX} temporarily to avoid the overwritten of TX data.

Regarding the input flow control unit, its logic design is basically the same as the output counterpart. In particular, a couple of input latches, $L1_{RX}$ and $L2_{RX}$, are employed for the ME-latching synchronization. As addressed in Chapter 3, this is required for the loosely-coupled data-link design to avoid synchronization failure. An input valid signal can be derived according to *rx_te* and *rx_ta*, as illustrated in Figure 5.3. Different from the output flow control, no extra latch is required for continuously receiving data in the RX FSM: it is intrinsically supported.

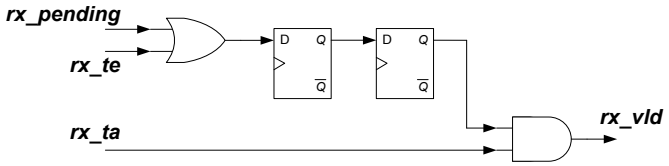


Figure 5.3 Input valid (high active) generation

As an example, Figure 5.4 presents the simulation waveforms of the major signals in the loosely-coupled data-link design. Also specified there are the timing parameters critical for performance. Since $T_{RX} > T_{TX}$, the RX-limited communication scenario was addressed. Both *tx_te* and *rx_re* were tied high, leading to a continuous transfer of data between the TX and the RX. The first eight data items are shown there, because of the lack of space. The received data of the RX input register FF_{RX} are highlighted in green. It can be seen that the data link achieved the maximum throughput of one datum per RX clock cycle. On the other side, the sending data of the TX output register FF_{TX}, as depicted in blue, turned out to be quite irregular in timing. As long as *tx_ta* remained high, the data were sent continuously. Once a transfer was pending, *tx_ta* became low, and an additional TX clock cycle was occupied by the data item.

5.1.3 Timing Constraints

First, consider the ME-latching scheme on the RX input data. It was analyzed in Chapter 3, under an assumption of zero delays on *ip_ai* and *ip_gi* from the MUTEX to $L1_{RX}$ and $L2_{RX}$. Since *ip_ai* and *ip_gi* are employed to gate the data latches, clock-tree synthesis is typically applied on both during layout. Taking the buffer-tree delays, d_{ip_ai} and d_{ip_gi}, into accunt, we have to ensure that *ip_ai* and *ip_gi* remain mutually exclusive when arriving at the latches. This in fact gives rise to the constraint of delay balancing between two signals:

$$d_{ip\_ai} = d_{ip\_gi}\,. \tag{5.1}$$

As an input to the RX flow control unit (IFC), the acknowledge signal *ip_ta* from the IPC is activated, however, independently of the local clock *rx_clk*. This necessitates a latch on *ip_ta* to align its transitions with the rising edges of *ip_gi*. The latched signal *ip_ta_l* goes through the XNOR and the MUX gates, then getting sampled by the *ip_te* flipflop at the rising edges of *rx_clk*. This addresses the critical path driven by *ip_ta_l*. The propagation delay, marked by d_{IFC}, determines the time interval from enabling the latch to triggering the flipflop without causing setup-time violations. On the other side, to avoid synchronization failures on the RX input register FF_{RX}, it is permitted to sample data only during the inactive phase of $L2_{RX}$ (see Chapter 3). Thus a double-side constraint is imposed on the RX clock-tree delay Δ_{CT}^{RX}:

$$d_{ip\_gi} + d_{IFC} < \Delta_{CT}^{RX} < d_{ip\_gi} + w_{MUTEX}^{RX}\,. \tag{5.2}$$

For the TX clock-tree delay Δ_{CT}^{TX}, it suffers from a similar constraint of (5.3). Each time launching a new transfer, data from the TX FSM propagates through FF_{TX} and is latched by L_{TX}. This causes another critical path in the output flow control: L_{TX} has to keep transparent until FF_{TX} being refreshed to avoid data loss. A short delay line can be inserted on *ip_te* for this purpose, as illustrated in Figure 5.1.

$$d_{op\_gi} + d_{OFC} < \Delta_{CT}^{TX} < d_{op\_gi} + w_{MUTEX}^{TX}\,. \tag{5.3}$$

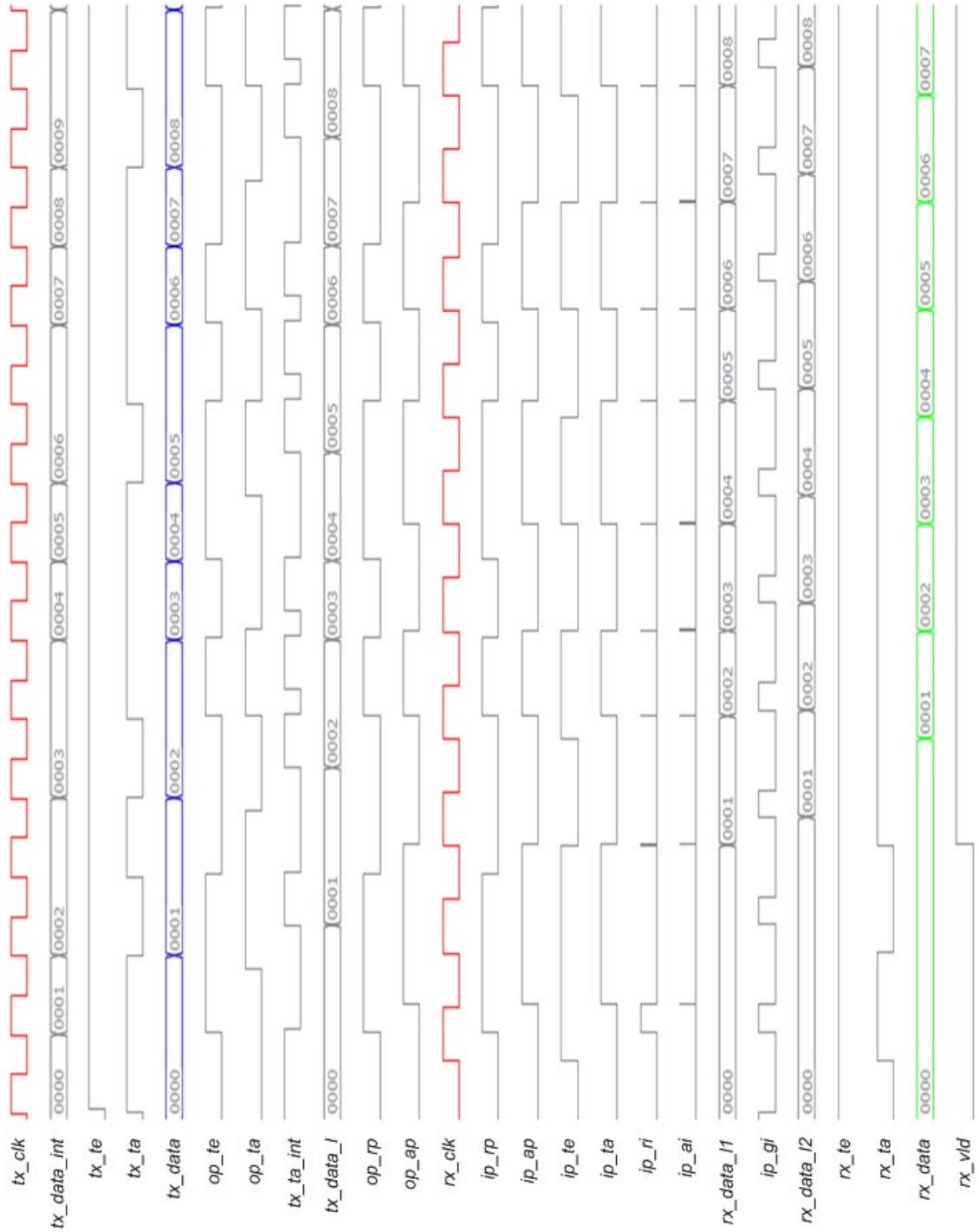


Figure 5.4 Simulation waveforms of the loosely-coupled data link at $T_{TX} \approx 6.1\ ns, T_{RX} \approx 8.3\ ns, w_{MUTEX}^{TX} = T_{TX}/2, w_{MUTEX}^{RX} = 3T_{RX}/4,$ $\Delta_{CT}^{TX} \approx 1ns, \Delta_{CT}^{RX} \approx 4ns, \Delta_{INT} = 0ns.$

5.1.4 Hardware Overheads

Except for the TX output register FF_{TX} and the RX input register FF_{RX}, which are highlighted by the red shadows, all the other circuits in Figure 5.1 are GALS specific. In comparison with the fully synchronous design, they account for the hardware overheads of the loosely-coupled data link. Among them, the asynchronous port controllers actually contribute the least, due to the simple gate-level circuits as shown in Figure 4.8. The flow control units are typically negligible as well: only three flipflops and one latch are utilized on each side. By contrast, the intermediate data latches and the on-chip ring oscillators dominate the hardware cost of the loosely-coupled data-link design. Given a data bus of N bits and a delay line with M slices, we can further estimate the data-link overhead O_{DL} by (5.4). There, O_{slice} and O_{latch} stand for the costs introduced by signal latch and single delay slice, respectively, either on power or on area depending on the different applications. An example of the gate-level implementation of programmable delay lines can be referred to Figure 2.7. For the large-scale systems with moderate-to-coarse grained partitioning, the loosely-coupled data-link design normally introduces marginal hardware overheads only. We will readdress this issue in Chapter 8 in great deal with experimental results.

$$O_{DL} \approx (M_{TX} + M_{RX}) \cdot O_{slice} + 3N \cdot O_{latch} . \tag{5.4}$$

5.2 Low-Overhead Design of Tightly-Coupled Data Link

Under the condition that continuous data transfer is not of the primary concern, we can simplify the data-link design by using the tightly-coupled I/O ports. The sequential handshaking between the OPC and the IPC, specified by STGs in Figure 4.5, indicates that the TX output data will be kept stable until being sampled by the RX clock. For this reason, the intermediate input latches, $L1_{RX}$ and $L2_{RX}$, are unnecessary on the RX side: data consistency is guaranteed by the tight coupling of I/O ports. Moreover, under the condition that the TX module never has to send data continuously, the intermediate output latch L_{TX} on the TX side also can be omitted. All these account for a low-overhead design of the tightly-coupled data link, as presented in Figure 5.5. Note that the input flow control is slightly different from in the loosely-coupled data link. The RX register FF_{RX} is allowed for data sampling only if a new item of transfer is valid on the datapath, which is signified by $ip\_ta\_l \neq ip\_te$. The design of TX and RX core modules follows the same schemes as depicted in Figure 4.3. The aforementioned timing constraints on the clock-tree delays are applicable as well. The proposed tightly-coupled data link can be modeled by (5.5) in terms of the hardware overheads. Only the ring oscillators have to be taken into account. It presents a rather lightweight design solution, adapted for the case where no intensive communication is required by the data link.

$$O_{DL} \approx (M_{TX} + M_{RX}) \cdot O_{slice} . \tag{5.5}$$

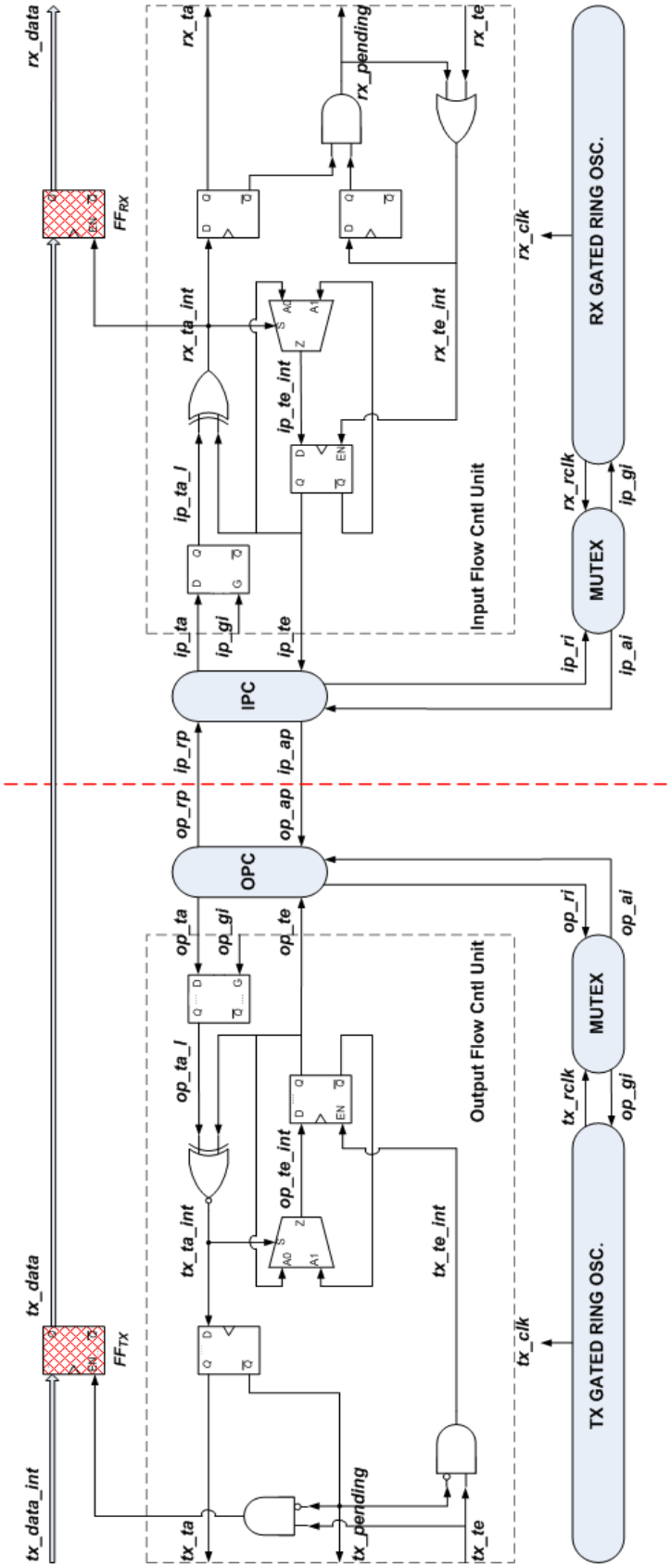


Figure 5.5 Low-Overhead design of the tightly-coupled GALS data link

5.3 Summary

This chapter has detailed the circuit design of GALS data links based on pausible clocking. Two representative solutions are proposed: the high-throughput design of the loosely-coupled data link and the low-overhead design of the tightly-coupled data link. Note that, for continuous data transfer, only three stages of latches are employed by the loosely-coupled data link. This addresses a remarkable advantage in terms of hardware overheads over the most dual-clock FIFO based designs of GALS data link.

We restrict our discussion on the point-to-point (P2P) data-link design. In theory, any system can be GALS adapted by using P2P interconnects only. Nevertheless, multi-port data exchange is a necessity of SOC designs with modularity and extendibility. An example of the ring interconnection based on pausible clocking was presented in [54]. Our GALS data-link design also can be generalized for multi-point communications. The details on this topic, however, go beyond the scope of this chapter.

Chapter 6

Power-Balanced System Partitioning

Despite the well-developed data-link circuits, system partitioning remains a critical aspect for GALS design. It defines how we can benefit most from dividing a system into a number of individual clock domains. Functional partitioning often sets a ground rule. It favors avoiding frequent communications across clock boundaries, which can sacrifice performance. Nevertheless, the criteria of optimal partitioning are essentially application dependent. The work in [55] presents an example on this issue, where GALS design was employed to enhance the security of cryptographic devices against side channel attacks. Intensive asynchronous communication turned out to be beneficial for introducing delay uncertainty on the cryptographic operations.

In this chapter we will focus on lowering the digital switching noise by using GALS design. Power-consumption balanced system partitioning is proposed for the first time. An in-depth systematic study is presented to address the advantages of power balanced GALS design over the existing low-noise synchronous design techniques.

6.1 Digital Switching Current

CMOS gates draw current flowing through the power distribution networks (PDNs) when switching states. The current is charging or discharging the capacitive loads on the internal and external nodes of gates. In a synchronous circuit, numerous logic gates can switch simultaneously at the edges of a global clock signal, leading to substantial current surges on the PDNs. With the shrinking feature size of MOS transistors, both integration density and working frequency of digital circuits are increased dramatically. This further increases the switching current.

The switching current in fact accounts for the primary source for digital switching noise. Due to the parasitic RLC impedance of the die/package/board-PDNs, a substantial VDD/GND voltage fluctuation, named digital power supply noise, can be introduced by the switching current [56]. The average on-chip power noise, rather than the peak noise, has been found to give rise to the timing variations on data paths. The noisy current also gets its way to be propagated into the substrate, via the parasitic p-n junction capacitance and the substrate/well contact resistance. The substrate potential is impacted, resulting in the “substrate noise”, well known for mixed-signal designers [57]. Moreover, undesired electromagnetic radiations could be produced, when the switching current flows through the parasitic antennas of the on-chip/board PDNs. This significantly affects the electro-magnetic compatibility (EMC) issues in digital systems [58].

6.2 Triangular Model of Switching Current

The switching current of a digital circuit is essentially determined by three factors: the input activity, the circuit topology and the current characteristics of each gate. However, due to the huge amount of logic cells, an accurate modeling of the current profiles for each of clock cycles turns out to be very challenging for nowadays large-scale digital design. Practically, we can approximate the switching current by a periodic current pulse in each cycle, which represents the ensemble average throughout time, superimposed by a noise signal from the cycle-to-cycle current variations [59]. In the frequency domain, the periodic component is found to account for the outstanding spectral peaks, often tens of dB above the noise floor. This actually indicated its dominance in the power of digital switching current. Measurements on typical medium-scale (40 k-gates) DSP modules demonstrated that the triangular waveform can be adopted as a first-order approximation of the average current profile [60] [61]. The piecewise-linear pattern is applicable as well, even if more details need to be covered. Recent work in [62] [63] explored the cycle-based modeling of digital switching current by stochastic analysis.

A triangular model of the digital switching current is illustrated in Figure 6.1. As a periodic signal, $i(t)$ has the same frequency f_0 as the clock signal. It is characterized by three parameters in each cycle: the peak value Ip, the rise time tr and the fall time tf. In addition, the total charge drawn through the power supply within each clock cycle can be obtained as the area of the triangular pulse, that is $Q = Ip(tr + tf)/2$.

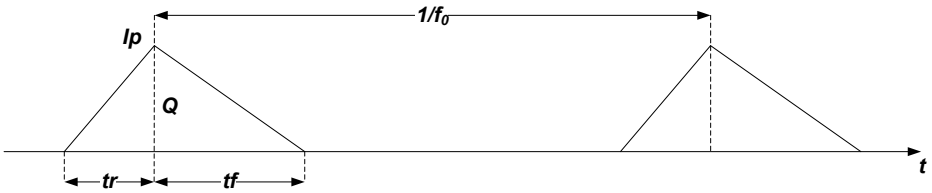


Figure 6.1 Triangular model of digital switching current

The periodic function approximation leads to a discrete spectrum of $i(t)$. It consists of the frequency components only at the harmonics of the clock frequency $f = nf_0$:

$$\mathbb{FS}(i(t)):\ i(t) = \sum_{n=-\infty}^{+\infty} \mathbb{F}(nf_0) \cdot e^{j2\pi nf_0 t}\ . \tag{6.1}$$

When the above triangular model is considered, the n^{th} spectral tone of $i(t)$, i.e., $F(nf_0)$ can be formulated as (6.2). Essentially, (6.2) presents a frequency-domain model of $i(t)$. Based on it, we will address the spectrum analysis and optimizations of digital switching current in the following sections.

$$\mathbb{F}(nf_0) = \frac{Ip}{j2\pi n}(\operatorname{sinc}\pi nf_0 tr - \operatorname{sinc}\pi nf_0 tf \cdot e^{-j\pi nf_0(tr+tf)}) \cdot e^{-j\pi nf_0 tr}\ . \tag{6.2}$$

6.3 Low-Noise Synchronous Design Techniques

Before digging into the details of low-noise GALS design, it is necessary to have an overview of the representative synchronous solutions reported in the literature. Inserting decoupling capacitors (decap) accounts for a well-developed approach among them [64]. On-chip decaps can offer transient current to the neighbor switching logic, thus lowering the supply current drawn on the PDNs. The efficiency, however, relies on several critical factors, such as the size, placement and recharging time of on-chip decaps. Remarkable overheads, both in area and in leakage power, could be introduced as a result. For large digital circuits this solution is therefore expensive. Alternatively, below we would like to emphasize on two low-cost design techniques, named supply current shaping and spread spectrum clocking, respectively. Note that, while still in the realm of synchronous design, both techniques have introduced clock modulation, either on phase or on frequency, to alleviate the digital switching noise.

6.3.1 Supply Current Shaping

The switching current surges can be efficiently smoothed by exploiting clock skew, i.e., to spread the switching activity in a digital circuit. Usually, it is achieved at a coarse level of granularity by dividing an entire system into a number of blocks. A global clock is synthesized, but with inter-block clock skew inserted deliberately. Phase modulation is thus performed on the clock signal. The switching activity in individual blocks is de-synchronized over time, and the current pulse in each clock cycle is spread, leading to a decrease on pulse peak and an increase both on rise and fall time. Many algorithms were proposed to optimize the logic grouping and the skew scheduling [61] [65]. The latest progress on this subject can be referred to [66], where the authors suggested clustering the combinational gates based on the path delay slacks with balanced load.

An in-depth spectrum analysis regarding the supply current shaping technique was first addressed by M. Badaroglu *et al.* in [60] [61]. Its efficiency on the power spectrum reduction of $i(t)$ was outlined, however, with a focus on the notch frequency allocation. Actually, there remains a fundamental issue to be solved, namely, what is the maximum attenuation we could expect for supply current shaping in practice. The power of $i(t)$ at each harmonic frequency is measured by the amplitude spectrum $a(nf_0)$ of (6.3). Given two particular scenarios, as considered below, (6.3) can be further simplified, leading to the following analytical study on the spectrum optimization of $i(t)$. It is of importance to note that, clock skew insertion typically comes with negligible overhead on the total charge consumed in each clock cycle [60] [61] [65]. In other words Q is kept constant, regardless of the current shape. This essentially accounts for a boundary condition to be respected in our analysis.

$$|\mathbb{F}(nf_0)| = a(nf_0) = \frac{Ip}{2\pi n} |\operatorname{sinc}\pi nf_0 tr - \operatorname{sinc}\pi nf_0 tf \cdot e^{-j\pi nf_0(tr+tf)}|. \qquad (6.3)$$

A. High-Frequency Harmonics

First, consider the higher order harmonics above the upper corner frequency of $i(t)$:

$$nf_0 > \max(\frac{1}{\pi tr}, \frac{1}{\pi tf}) . \tag{6.4}$$

The *sinc* function in (6.3) can be efficiently simplified under this condition:

$$a(nf_0) \approx \frac{Ip}{2\pi n} | \frac{1}{\pi nf_0 tr} - \frac{1}{\pi nf_0 tf} \cdot e^{-j\pi nf_0(tr+tf)} | . \tag{6.5}$$

An upper bound of $a(nf_0)$, denoted by $\tilde{a}(nf_0)$, can be thus derived:

$$a(nf_0) \le \frac{Ip}{2\pi n}(\frac{1}{\pi nf_0 tr} + \frac{1}{\pi nf_0 tf}) = \frac{Q}{(\pi n)^2 f_0} \cdot \frac{1}{tr \cdot tf} = \tilde{a}(nf_0) . \tag{6.6}$$

Then define the relative pulse width λ of $i(t)$ as follows:

$$0 < \lambda = (tr + tf) \cdot f_0 \le 1 . \tag{6.7}$$

Given a relative width, (6.8) always holds:

$$tr \cdot tf \le \frac{1}{4}(tr + tf)^2 = (\frac{\lambda}{2f_0})^2 . \tag{6.8}$$

Taking (6.8) into (6.6), we have:

$$\tilde{a}(nf_0) \ge \frac{4Qf_0}{(\pi n\lambda)^2} . \tag{6.9}$$

The equality holds iff $tr = tf$. Actually (6.9) presents a quantitative measure on the efficiency of current shaping in terms of lowering the high-frequency harmonics. With an increasing relative width λ, the upper bound $\tilde{a}(nf_0)$ of the amplitude spectrum drops at a ratio of $1/\lambda^2$, i.e. -40 dB per decade. It reaches a minimum of (6.10) at $\lambda = 1$, which indicates the theoretical limit on spectrum reduction by current spreading.

$$\min(\tilde{a}(nf_0)) = \frac{4Qf_0}{(\pi n)^2} . \tag{6.10}$$

B. Low-Frequency Harmonics

In contrast, consider the harmonics below the lower corner frequency of $i(t)$:

$$nf_0 < \min(\frac{1}{\pi tr}, \frac{1}{\pi tf}) . \tag{6.11}$$

The *sinc* function in (6.3) can be well approximated by 1 in this case, leading to a rather simple expression of $a(nf_0)$ as shown in (6.12). The lower order spectral peaks turn out to be determined only by the total charge consumed in each clock cycle. For this reason, we would benefit little from the supply current shaping at the low-frequency harmonics. This fundamentally impacts its application.

$$a(nf_0) \approx \frac{Ip}{2\pi n} | 1 - e^{-j\pi nf_0(tr+tf)} | = Qf_0 | \operatorname{sinc} \pi nf_0 \frac{1}{2}(tr + tf) | \approx Qf_0 . \tag{6.12}$$

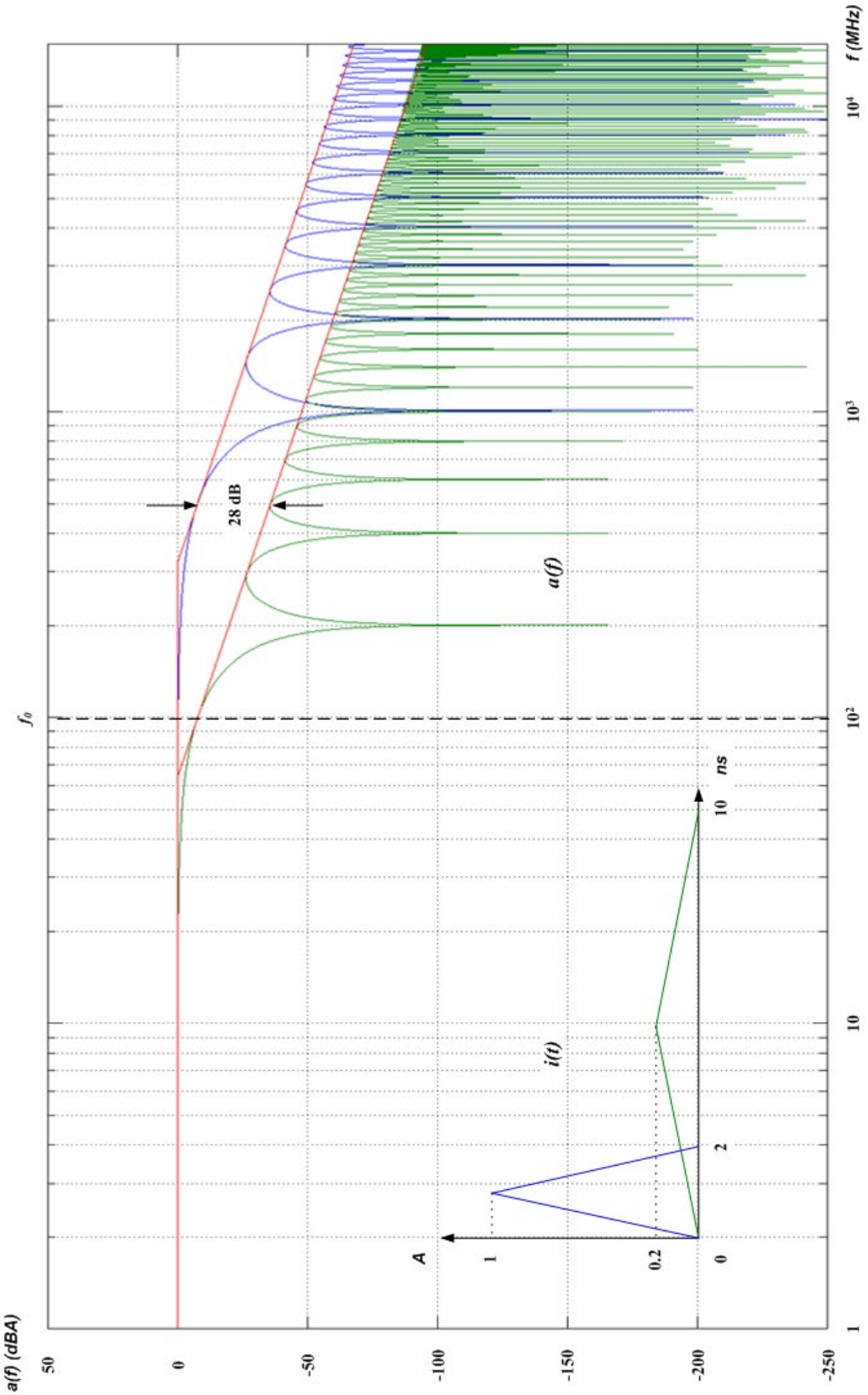


Figure 6.2 Spectrum optimization by supply current shaping – an example

C. An Example

Figure 6.2 graphically addresses our above analysis. Two triangular current waveforms are taken into consideration. Each of them represents a typical scenario of supply current shaping: a barely spread current pulse with $\lambda = 0.2$, as depicted in blue, versus a sufficiently spread current pulse with $\lambda = 1$, plotted in green. Without loss of generality, symmetric waveforms, which have $tr = tf$, are applied. For clarity of comparison, there we show the spectrum envelop $a(f)$, defined by each of the single-cycle current pulses, instead of the discrete spectrum. The spreading of supply current is found to account for a dramatic attenuation of the spectral peaks in the high-frequency domain. A down shift of 28 dB has been measured on the upper bounds of the corresponding spectra, as the red lines highlighted. It matches our analytical estimation of (6.9) very well. On the other side, however, at the fundamental clock frequency of $f_0 = 100\ MHz$, only an approx. 7 dB drop can be gained instead. Actually, spreading the current waveform over an entire clock cycle, as assumed in this example, might be overoptimistic, if not impossible. In most cases, the maximum clock skew accommodated by a synchronous circuit is tightly restricted due to the setup/hold-time constraints. This significantly limits the efficiency of current spreading. It is our opinion that, especially when the lower harmonics are of the concern, only a marginal benefit, if any at all, could be achieved.

6.3.2 Spread Spectrum Clocking

Even with clock skew insertion, the switching activity in individual clock domains are still synchronized by the global clock. The power of $i(t)$ is concentrated at the clock harmonics as a result, producing strong spectral peaks in the frequency domain. Actually further attenuation of the spectral peaks of $i(t)$ can be obtained by modulating the clock frequency. The power at each of clock harmonics will be spread over a wide bandwidth as a result, leading to the lower spectral peak. This accounts for the principle of spread spectrum clocking. It was first suggested in [67] for reducing the radiated emissions of digital systems, and then applied to address the power supply noise [60]. Traditionally, periodic frequency modulation is applied for the spread spectrum clock generation [68] [69] [70]. A recent study in [71] explored to maximize the power attenuation by chaotic-PAM modulation.

The modulation index β is defined by (6.13), where Δf denotes the peak frequency deviation from the central frequency f_0, and f_m the modulation waveform frequency:

$$\beta = \frac{\Delta f}{f_m}. \tag{6.13}$$

According to Carson's law [72], the bandwidth at the n^{th} harmonic after frequency modulation can be approximated as follows:

$$BW(nf_0) \approx 2(n\beta + 1) \cdot f_m. \tag{6.14}$$

Note that frequency modulation doesn't change the total spectral power surrounding each clock harmonic. Given a modulation index, we can therefore obtain the maximum spectral peak attenuation of (6.15) by evenly distributing power over side lobes:

$$A_{dB}(nf_0) \approx 10\log 2(n\beta + 1)\,. \tag{6.15}$$

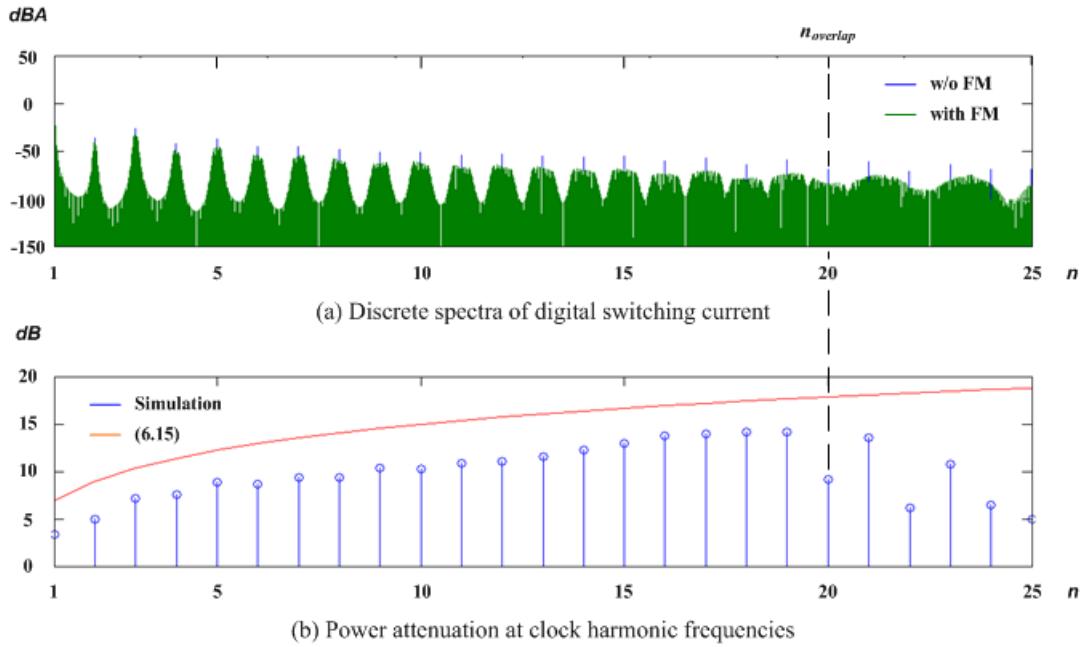
The above equation (6.15) indicates that the modulation efficiency is determined by $n\beta$ at the n^{th} harmonic. However, there are two particular scenarios to be mentioned. In the case of $n\beta < 1$, spectrum spreading at the central harmonic nf_0 is negligible in practice, with a power attenuation of virtually zero. For this reason we can specify the cutoff harmonic index n_{cutoff} as (6.16):

$$n_{cutoff} \cdot \beta = 1 \Leftrightarrow n_{cutoff} = \frac{1}{\beta}\,. \tag{6.16}$$

On the other side, if $n\beta$ gets overly large, the spread spectra between the adjacent harmonics will be overlapped. The power attenuation at nf_0 is sacrificed consequently. This accounts for the overlapping harmonics index $n_{overlap}$ derived by (6.17):

$$(n_{overlap} \cdot \beta + 1)f_m + ((n_{overlap} + 1)\beta + 1)f_m = f_0 \Leftrightarrow n_{overlap} = \frac{1}{\beta}\left(\frac{f_0}{2f_m} - 1\right) - \frac{1}{2}\,. \tag{6.17}$$

Figure 6.3 draws an example of spectrum spreading by frequency modulation. A triangular modulation scheme, with a modulation index of $\beta \approx 1.5$ ($f_0 = 100MHz, \Delta f = 2.3\%f_0$ and $f_m = f_0/64$), is taken into account. Remarkable power attenuation has been gained at the clock harmonic frequencies. Our analytical estimation of (6.15) reasonably matches the simulation results for $n \leq 19$. From the 20th harmonic on, spectrum overlapping is incurred, which impacts the efficiency of power reduction.



(a) Discrete spectra of digital switching current

(b) Power attenuation at clock harmonic frequencies

Figure 6.3 Spectrum spreading by clock frequency modulation

According to the above analysis, we can specify the major design criteria for spread spectrum clocking in two aspects: maximizing $A_{dB}(nf_0)$ on one side, while decreasing n_{cutoff} and increasing $n_{overlap}$ on another side. Again, take the triangular modulation waveform for example. It can be characterized by two parameters, as depicted in Figure 6.4: the modulation granularity ΔT on clock period and the cycle number N_m applied to a modulation waveform.

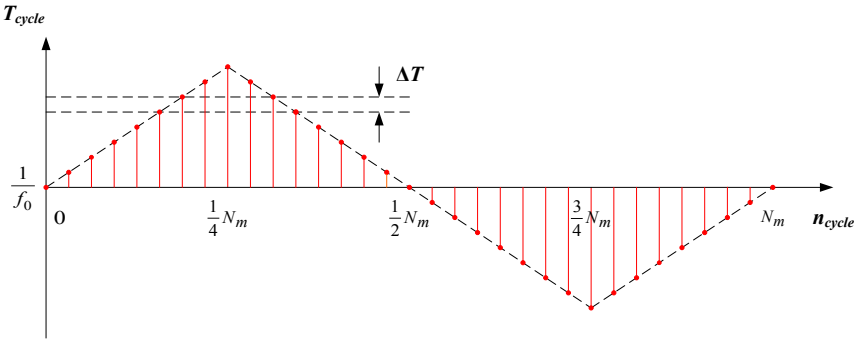


Figure 6.4 Triangular modulation waveform – an example

Further define the peak frequency deviation ratio α as (6.18):

$$\alpha = \frac{\Delta f}{f_0}. \tag{6.18}$$

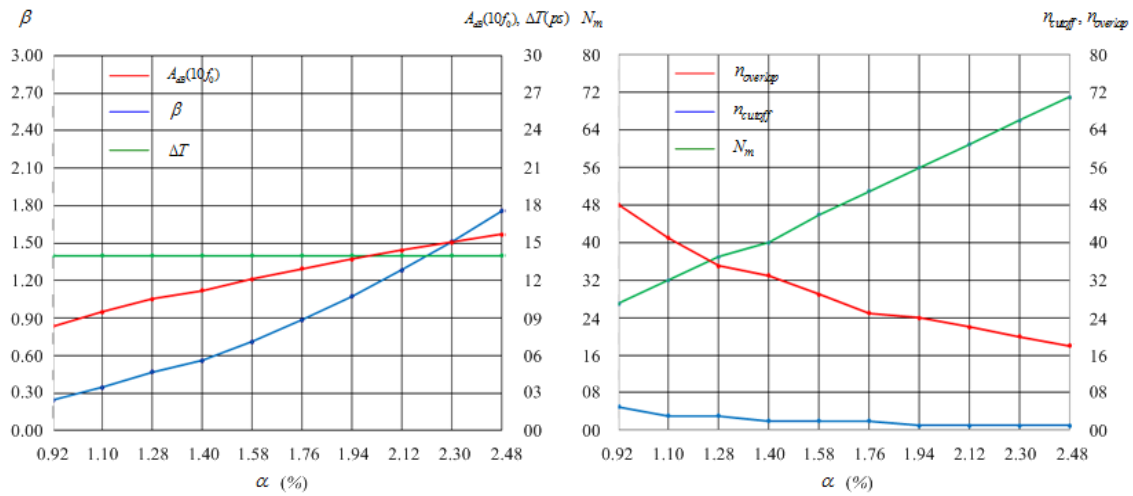
In practice α is often constrained by $\alpha \ll 1$. That is, only tiny frequency modulation is allowed. Under this condition, it can be expressed based on ΔT and N_m:

$$\alpha \approx \frac{1}{4} N_m \cdot \Delta T \cdot f_0 . \tag{6.19}$$

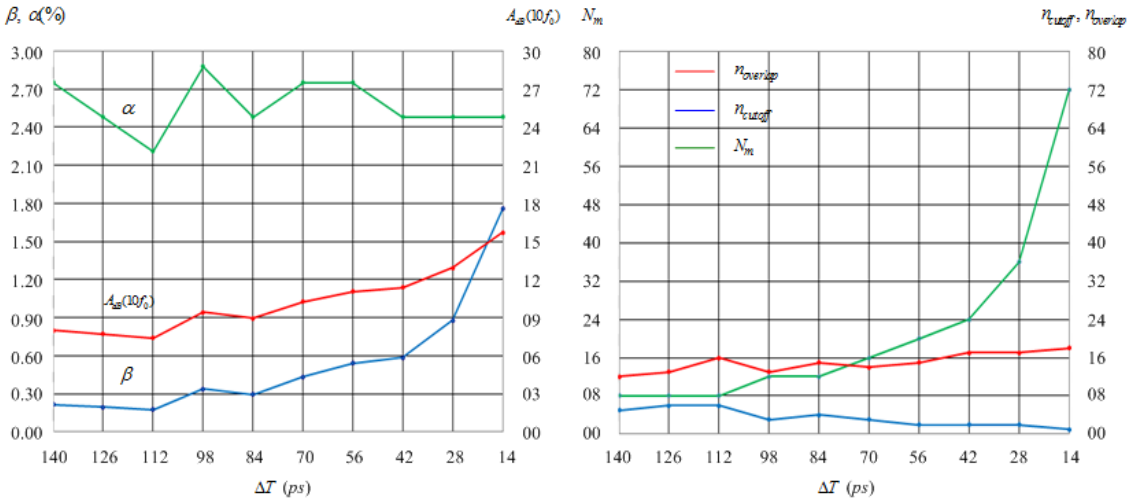
Substituting (6.18) and (6.19) into (6.13), also noticing that $N_m = f_0 / f_m$, we can derive the frequency modulation index as (6.20). That is, β is directly proportional to the square of the frequency deviation ratio α, and inversely proportional to the modulation granularity ΔT.

$$\beta \approx \frac{4\alpha^2}{\Delta T \cdot f_0}. \tag{6.20}$$

Now taking (6.20) into (6.15), (6.16) and (6.17), we are able to formulate $A_{dB}(nf_0)$, n_{cutoff} and $n_{overlap}$ all in terms of α and ΔT. The analytical expressions, however, are omitted here for the lack of space. Figure 6.5 graphically shows examples of the design optimization of spread spectrum clocking. The following can be further concluded. First, given ΔT, increasing α can efficiently improve $A_{dB}(nf_0)$, while with a dramatic drop on $n_{overlap}$. This addresses an important fact, namely, a large frequency deviation actually may not improve the spectrum spreading. Second, given α in contrast, reducing ΔT will contribute to increase both $A_{dB}(nf_0)$ and $n_{overlap}$ as well. To minimize the modulation granularity is of particular significance for this reason.



(a) Given a modulation granularity ΔT, increasing frequency deviation ratio α



(b) Given a frequency deviation ratio α, decreasing modulation granularity ΔT

Figure 6.5 Comparisons in design optimization of spread spectrum clocking

6.3.3 Further Discussions

Note that the lower order harmonics of the clock frequency typically dominate the AC power of digital switching current. It is especially the case for the fundamental tone, which often has the highest power and, therefore, accounts for the strongest noise source. As a result, the attenuation of spectral peaks at the low-frequency harmonics deserves our particular attention. We have proven that, at the lower order harmonics, little power reduction can be gained by supply current shaping (see (6.12)). To achieve a sufficient attenuation at the clock harmonics with low indexes is also rather challenging for spread spectrum clocking, because its efficiency is logarithmically linear to $n\beta$ in general (see (6.15)). On the other side, we see that both the techniques are actually effective to drop the spectral peaks at high-frequency harmonics. Taking all these issues into account, we will address the alternative GALS design solutions, with a focus on power reduction of switching current at lower order harmonics.

6.4 Plesiochronous Design with Power Balanced Partitioning

6.4.1 Current Spectrum in GALS Design

Consider a digital system that is partitioned into a total of *M* clock domains. Within each particular clock domain D_m, the gates are triggered by a local clock at an individual frequency f_m. For the theoretical rigor in the following analysis, we assume the system being rationally clocked. That is, given any pair of D_i and D_j, the frequency ratio f_i/f_j is a rational number:

$$\frac{f_i}{f_j} = \frac{I}{J}, \text{ where } I, J \in N, \text{ for } 1 \le \forall i, j \le M . \tag{6.21}$$

Denote the switching current from D_m as $i_m(t)$, and its frequency as f_m. According to (6.1), we can specify the discrete spectrum of $i_m(t)$ as:

$$\mathbb{FS}(i_m(t)): \; i_m(t) = \sum_{n=-\infty}^{+\infty} \mathbb{F}_m(nf_m) \cdot e^{j2\pi n f_m t} . \tag{6.22}$$

The switching current of a whole system is simply the sum of $i_m(t)$ from all the *M* clock domains. Under the condition of (6.21), it remains periodic, but with a frequency of $1/LCM(1/f_1, \ldots, 1/f_M)$, where *LCM* represents the least common multiple function. We can therefore, derive its discrete spectrum as the superposition of (6.22) based on the linear property of Fourier series:

$$\mathbb{FS}_G(\sum_m i_m(t)): \; \sum_m i_m(t) = \sum_m (\sum_{n=-\infty}^{+\infty} \mathbb{F}_m(nf_m) \cdot e^{j2\pi n f_m t}) . \tag{6.23}$$

Above (6.23) gives the spectrum of digital switching current for a GALS system. It consists of the harmonic components of all the *M* local clock frequencies. In contrast, by taking $f_m = f_0$ into (6.23), we can also get the current spectrum for the corresponding synchronous system. As seen in (6.24), it has spectral components only at the harmonics of the global clock frequency f_0, but each component combines the contributions of all the *M* clock domains.

$$\mathbb{FS}_S(\sum_m i_m(t)): \; \sum_m i_m(t) = \sum_{n=-\infty}^{+\infty} (\sum_m \mathbb{F}_m(nf_0)) \cdot e^{j2\pi n f_0 t} . \tag{6.24}$$

It is noteworthy that both (6.23) and (6.24) actually represent the current spectra for the same system, just with different clocking strategies. Obviously, by desynchronizing the clocks, GALS design allows much more flexibility for shaping the current spectrum than its synchronous counterpart. The working frequencies of *M* local clocks span a vast design space for the spectrum optimization. However, in comparison with the fully synchronous design, clock de-synchronization also gives rise to some overheads in throughput, power and area, if not properly addressed. Therefore, the frequency planning of a GALS system is in principle a trade-off between switching noise reduction, performance and hardware requirements. Practically it calls for a case-by-case study.

6.4.2 Spectrum Spreading by Plesiochronous Clocking

Taking into account the above analysis, we constrain our work on the GALS design with plesiochronous clocks. That is, any local clock works at a frequency f_m, which only has a slight mismatch from a central frequency f_0:

$$f_m = f_0 \cdot (1 + \varepsilon_m), \text{ where } | \varepsilon_m | << 1, \text{ for } 1 \le \forall m \le M \ . \tag{6.25}$$

For two issues the plesiochronous design deserves our attention. From the perspective of synchronous design, it actually presents "the most conservative" alternative of clock de-synchronization. The operating frequency and in turn the processing capability of each clock domain is maintained. Typically, it is possible to adapt a synchronous design to be plesiochronous without significantly sacrificing performance. It can be therefore applied as a systematic GALS solution for low-noise design. On the other side, plesiochronous clocking also accounts for "the most pessimistic" scheme of clock de-synchronization. The system is virtually synchronized at a global clock frequency. Our deviation thus will draw a lower bound of spectrum optimization by GALS design.

Substituting (6.25) into (6.23), we have the discrete spectrum of switching current for a plesiochronous system with *M* clocks, shown in (6.26). Surrounding the n^{th} central harmonic $f = nf_0$ are *M* spectral components, each departing from nf_0 in a distance of $\Delta f = \varepsilon_m \cdot nf_0$. Compared with (6.24), we see that converting a synchronous design to be plesiochronous leads to a spectrum spreading. At each harmonic frequency a single synchronous peak is replaced by *M* plesiochronous peaks. Fine grained partitioning with a larger *M* is generally beneficial for the spectrum spreading. Given an *M*, however, the number of plesiochronous peaks is constant at all the harmonics.

$$\mathbb{FS}_P(\sum_m i_m(t)) : \ \sum_m i_m(t) = \sum_{n=-\infty}^{+\infty} (\sum_m \mathbb{F}_m(nf_0(1+\varepsilon_m)) \cdot e^{j2\pi nf_0(1+\varepsilon_m)t}) . \tag{6.26}$$

We define the spectral peak attenuation $A_{dB}(nf_0)$ to be the amplitude ratio between the synchronous peak and the highest plesiochronous peak at the n^{th} harmonic frequency $f = nf_0$. It can be expressed as (6.27) based on (6.24) and (6.26). Essentially, $A_{dB}(nf_0)$ measures the power reduction at harmonic frequencies due to the spectrum spreading by plesiochronous design.

$$A_{dB}(nf_0) = 20\log \frac{|\sum_m \mathbb{F}_m(nf_0)|}{\max(|\mathbb{F}_m(nf_0(1+\varepsilon_m))|)} . \tag{6.27}$$

Formula (6.27) indicates that, to lower the spectrum power at the n^{th} harmonic, the superposition of spectral components from all the *M* clock domains has to be stronger than any single tone. With the triangular current model, the n^{th} spectral component of $i_m(t)$, $F_m(nf_m)$, can be specified according to (6.2):

$$\mathbb{F}_m(nf_m) = \frac{Ip_m}{j2\pi n}(\operatorname{sinc}\pi nf_m tr_m - \operatorname{sinc}\pi nf_m tf_m \cdot e^{-j\pi nf_m(tr_m+tf_m)}) \cdot e^{-j\pi nf_m tr_m} . \tag{6.28}$$

6.4.3 Attenuation at Lower Order Harmonics

Generally speaking, given all the $i_m(t)$ in a plesiochronous design, the spectral peak attenuation at any harmonic frequency can be derived by (6.27). However, as previously mentioned, we are interested in the lower order harmonics, which actually dominant the AC power of the switching current. Again consider the clock harmonics below the lower corner frequency of $i_m(t)$:

$$nf_m < \min(\frac{1}{\pi tr_m}, \frac{1}{\pi tf_m}). \tag{6.29}$$

Under the condition of (6.29), the *sinc* function in (6.28) can be well approximated by 1. This leads to a significant simplification on $F_m(nf_m)$:

$$\mathbb{F}_m(nf_m) \approx -j \cdot f_m \cdot Q_m \cdot e^{-j\pi nf_m(tr_m+(tr_m+tf_m)/2)}. \tag{6.30}$$

Ignoring the instant fluctuations on the on-chip power supply voltages, Q_m can be estimated based on the average power dissipation $\bar{P}_m$ of the clock domain:

$$Q_m = \int_0^{1/f_m} i_m(t)dt = \frac{\overline{i_m(t)}}{f_m} \approx \frac{\overline{P}_m}{VDD \cdot f_m}. \tag{6.31}$$

For the convenience of analysis, we redefine the relative width $\lambda_m$of the triangular current pulse $i_m(t)$ as (6.32). Similar to (6.7), it gives a measure of the duration of the switching current relative to the clock period.

$$\lambda_m = (tr_m + (tr_m + tf_m)/2) \cdot f_m. \tag{6.32}$$

In accordance with (6.31) and (6.32), (6.30) can be rewritten as:

$$\mathbb{F}_m(nf_m) = \frac{-j}{VDD} \cdot \overline{P}_m \cdot e^{-j\pi n\lambda_m}. \tag{6.33}$$

Substituting (6.33) to (6.27), an elegant expression of current spectrum attenuation at the low-frequency harmonics can be therefore derived:

$$A_{dB}(nf_0) = 20\log\frac{|\sum_m \overline{P}_m \cdot e^{-j\pi n\lambda_m}|}{\max(\overline{P}_m)}. \tag{6.34}$$

In essence (6.24) reveals three degrees of freedom offered by plesiochronous design for attenuating spectral peaks: *M*, the amount of clock domains; λ_m, the relative width of $i_m(t)$; and $\bar{P}_m$, the breakdown of average power over clock domains. Note that there is also a boundary condition imposed by the plesiochronous clocking to be respected. That is, if the switching activity of asynchronous interface circuits is marginal, which is often the case for the moderate-to-coarse grained GALS partitioning, a plesiochronous system consumes virtually the same power as its synchronous counterpart. For this reason, we have the constraint of (6.35) in our following analysis.

$$\sum_m \overline{P}_m \approx P_{total\_const} \tag{6.35}$$

6.4.4 Power-Consumption Balanced Partitioning

Formula (6.35) paves our way to the optimal partitioning of plesiochronous design for spectral peak reduction. Due to the lack of capability, if not feasibility, of accurately modeling the transient current waveforms, it seems neither appealing nor practical for us to take λ_m as a partitioning metric [73]. In contrast, the power consumption $\bar{P}_m$, even for large-scale systems, can be estimated by nowadays CAD tools very well. This accounts for our interest in the power based GALS partitioning.

Consider the simplest scheme of partitioning a system balanced in terms of power consumption, that is:

$$\overline{P}_m = \frac{P_{total\_const}}{M}, \text{ for } 1 \le \forall m \le M . \tag{6.36}$$

In this case, (6.34) can be simplified, leading to a very concise expression of (6.37). That is, the attenuation at the n^{th} harmonic is determined by the sum of M unit phasors, each having a phase proportional to λ_m:

$$A_{dB}(nf_0) = 20\log|\sum_m e^{-j\pi n\lambda_m}|. \tag{6.37}$$

The maximum of (6.37) is presented in (6.38). It implies a constant λ_m, and in turn an identical pulse width of $i_m(t)$, for all the M clock domains. Obviously, this is over-optimistic for most of the practical designs.

$$\begin{aligned} &\max(A_{dB}(nf_0)) = 20\log M \\ &\text{iff } \lambda_m = \lambda, \text{ for } 1 \le \forall m \le M \end{aligned} . \tag{6.38}$$

Instead, we can assume an individual λ_m for each of $i_m(t)$. Due to the condition of (6.36) on balanced power, this further implies a different Ip_m on each $i_m(t)$. Essentially, the ratio between the maximum and the minimum λ_m, denoted by Λ, measures the shape variations (or uncertainty) of switching current in the M clock domains:

$$\Lambda = \frac{\lambda_{\max}}{\lambda_{\min}} . \tag{6.39}$$

Without loss of generality, consider an even distribution of λ_m over $[\lambda_{min}, \lambda_{max}]$:

$$\lambda_m = \lambda_{\min} + \frac{\lambda_{\max} - \lambda_{\min}}{M-1}(m-1) . \tag{6.40}$$

Taking (6.39) and (6.40) into (6.37), we can thus derive an analytical expression of the spectral peak attenuation on the current shape variations in individual blocks:

$$A_{dB}(nf_0) = 20\log\left|\frac{1-e^{-j\frac{M}{M-1}\pi n\lambda_{\min}(\Lambda-1)}}{1-e^{-j\frac{1}{M-1}\pi n\lambda_{\min}(\Lambda-1)}}\right| . \tag{6.41}$$

Note that, according to (6.29), Λ is actually constrained by (6.42):

$$0 < \pi n\lambda_{\min}(\Lambda-1) < 2 . \tag{6.42}$$

Therefore, in the condition of $M \gg 1$, we have the Taylor series approximation of:

$$e^{-j\frac{1}{M-1}\pi n\lambda_{\min}(\Lambda-1)} \approx 1 + j\frac{\pi n\lambda_{\min}(\Lambda-1)}{M-1}. \tag{6.43}$$

Substituting (6.43) into (6.41), $A_{dB}(nf_0)$ can be simplified as follows:

$$\begin{aligned} A_{dB}(nf_0) &\approx 20\log M + o(\Lambda) \\ o(\Lambda) &= 20\log(\operatorname{sinc}\frac{1}{2}\pi n\lambda_{\min}(\Lambda-1)) \end{aligned}. \tag{6.44}$$

We get an offset term $o(\Lambda)$, introduced by the shape variations on $i_m(t)$, in addition to the theoretical maximum of $A_{dB}(nf_0)$. The magnitude of $o(\Lambda)$ defines the robustness of power balanced partitioning against current shape variations in terms of spectral peak attenuation. Under the constraint of (6.42), $o(\Lambda)$ is below zero, and reaches its minimum of -1.5 dB at $\Lambda = 1 + 2/\pi n\lambda_{min}$. This corresponds to the maximum loss on $A_{dB}(nf_0)$. For a large *M*, say $M > 6$, it is marginal ($< 10\%$) actually.

Taking into account the above discussion, we can reasonably omit $o(\Lambda)$ in (6.44) in the case of $M \gg 1$. This leads to our final expression of (6.45) regarding $A_{dB}(nf_0)$:

$$A_{dB}(nf_0) \approx 20\log M, \text{ for } M \gg 1. \tag{6.45}$$

It is revealed that, in particular at the lower order harmonics, the plesiochronous design with power balanced partitioning accounts for a spectral peak attenuation on switching current, which is logarithmically linear to the number of clock domains introduced in the system. Fine-grained partitioning is preferred as a result. In comparison with the spread spectrum clocking technique, it presents an efficient alternative to cope with noisy low-frequency harmonics. Also derived is its robustness against the current shape variations in individual blocks. This further denotes a significant advantage over the supply current shaping, especially when applied to large-scale system designs. The above two findings conclude our analytical study.

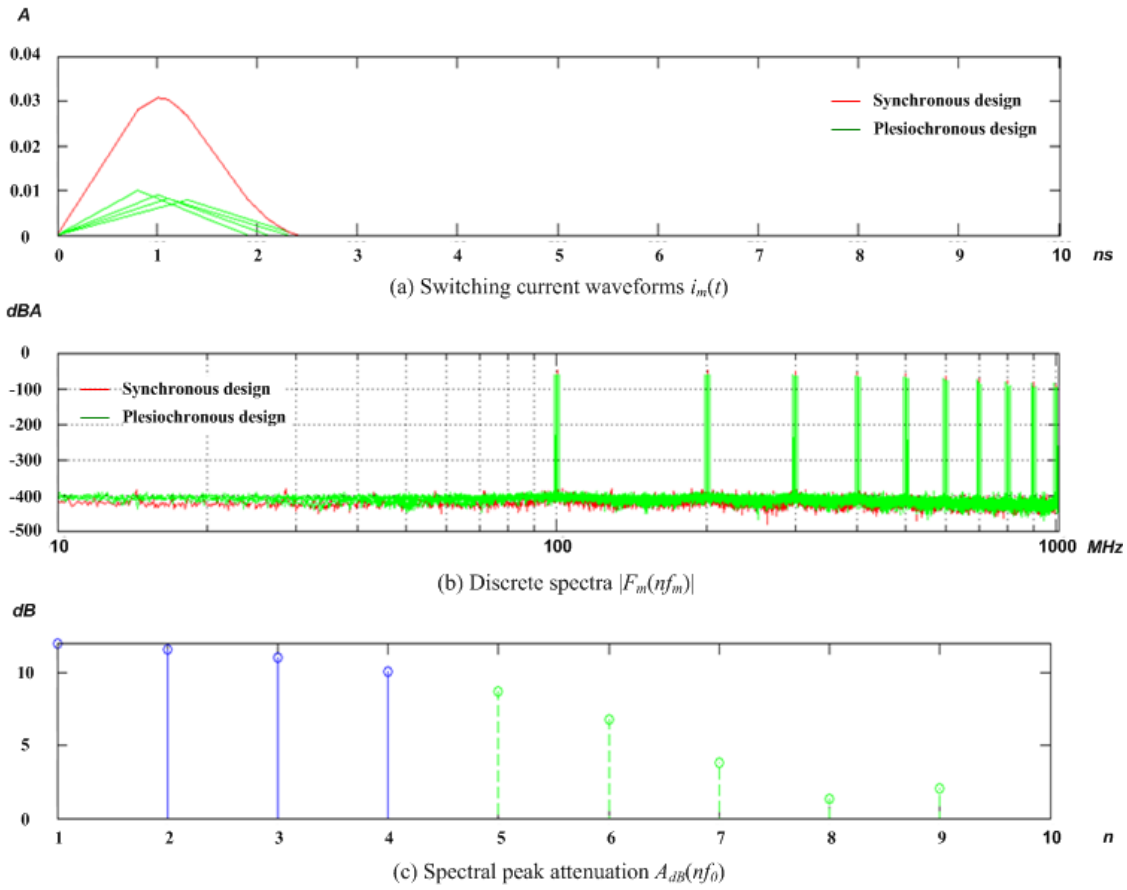
6.5 Numeric Simulations

Although above theoretical analysis seems complicated, the verification is straight-forward. A software tool for the spectrum analysis of digital switching current, based on *MATLAB*, has been developed. Given a particular system, we can specify the number of clock domains (*M*), the working frequencies (f_m) and the current pulse $i_m(t)$ on each of the clock domains. The triangular current model is applied, as plotted in Figure 6.1. On each $i_m(t)$ an FFT is performed to calculate the spectrum. According to (6.23), the sum of $i_m(t)$ spectra gives the current spectrum of the GALS system ($f_i \neq f_j$). In contrast, the FFT on the sum of $i_m(t)$ represents the current spectrum of the corresponding syn-chronous design ($f_i = f_j$). We can therefore address the spectral peak attenuation in the simulations. Appendix A provides our main source code for reference.

Figure 6.6 first illustrates an intuitive example on the current spectrum of a plesiochronous design. With $M = 4$, each harmonic peak of the synchronous baseline design is surrounded by four sub-peaks of the plesiochronous design. Due to the power balance in the clock domains, which is indicated by the same area covered by each of $i_m(t)$, a 12 dB reduction has been achieved on the first two clock harmonics. It perfectly matches our estimation of (6.45).

As shown in (6.34), the power breakdown in fact accounts for another dimension to be explored. Instead of the power-balanced partitioning, the power ratio P, as defined in (6.46), can be employed to measure the difference among clock domains in terms of the power dissipation. Together with the width ratio of current pulses Λ, a vast design space is thus provided for GALS partitioning. We have addressed this issue by comprehensive simulations. Figure 6.7 and 6.8 show the results for $M = 4$ and $M = 8$, respectively, at a sweep from 1 to 4 with a step of 0.15 on P and on Λ. The power balanced partitioning (the rightmost curves in each figure) manifests efficiency in both examples. In the case of $\Lambda \leq 2$, it is found to be the optimal scheme actually, giving the maximum attenuation at the fundamental clock frequency. Even for $2 < \Lambda < 4$, its results are still comparable with the best.

$$P = \frac{\overline{P}_{\max}}{\overline{P}_{\min}}. \tag{6.46}$$

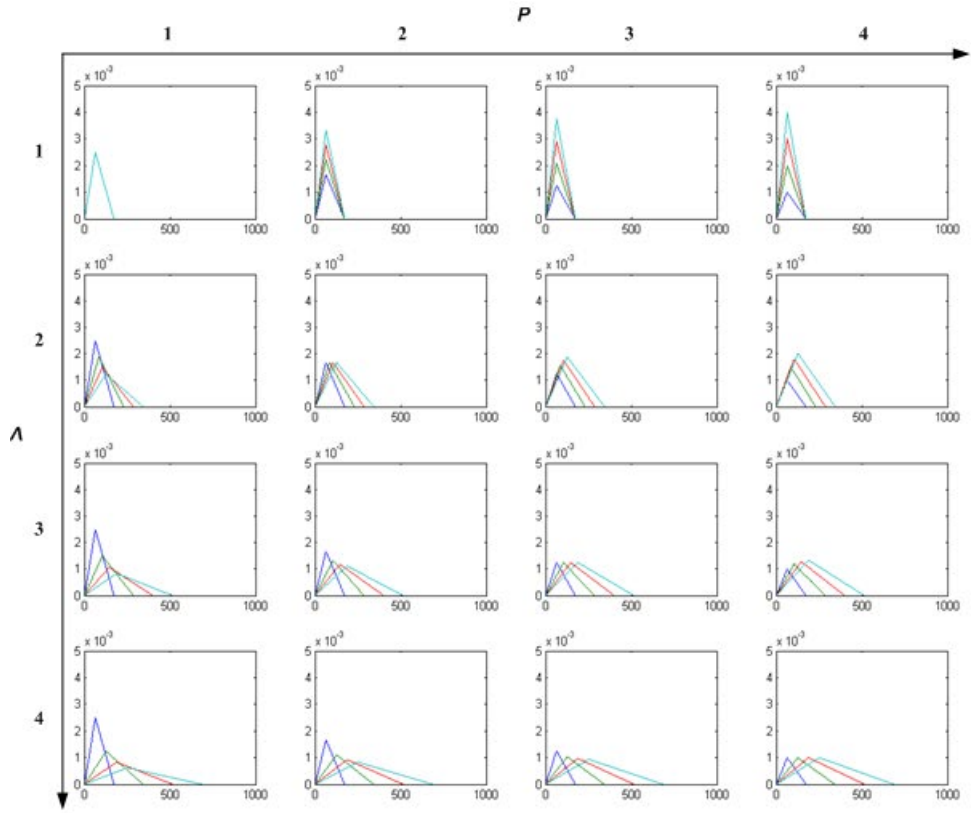


(a) Switching current waveforms $i_m(t)$

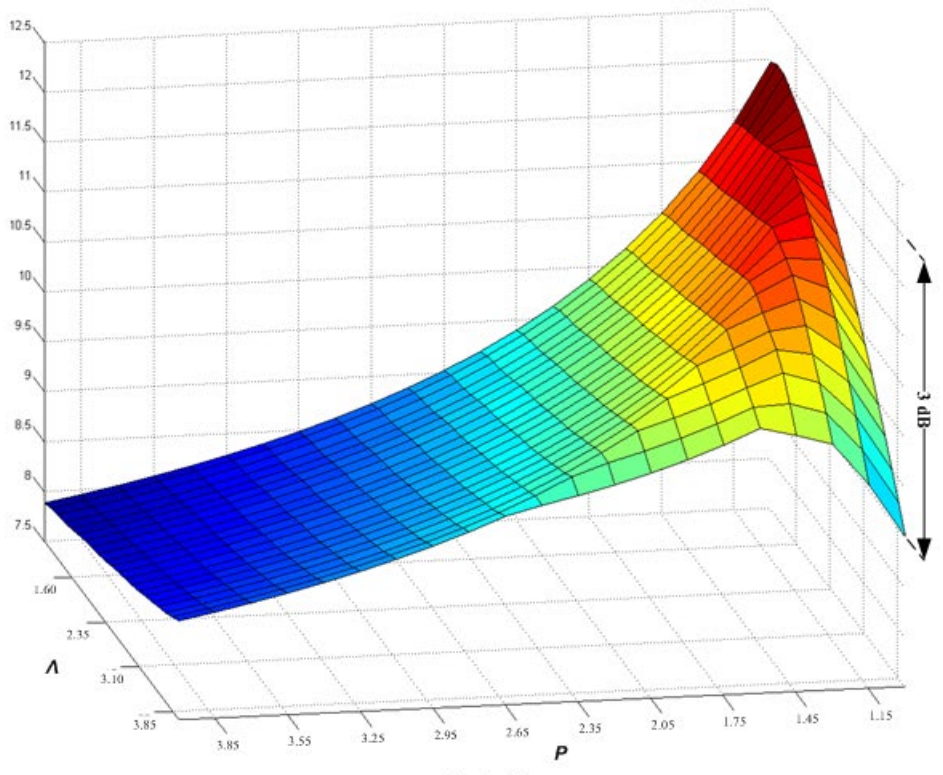
(b) Discrete spectra $|F_m(nf_m)|$

(c) Spectral peak attenuation $A_{dB}(nf_0)$

Figure 6.6 Current spectra of power-balanced plesiochronous design

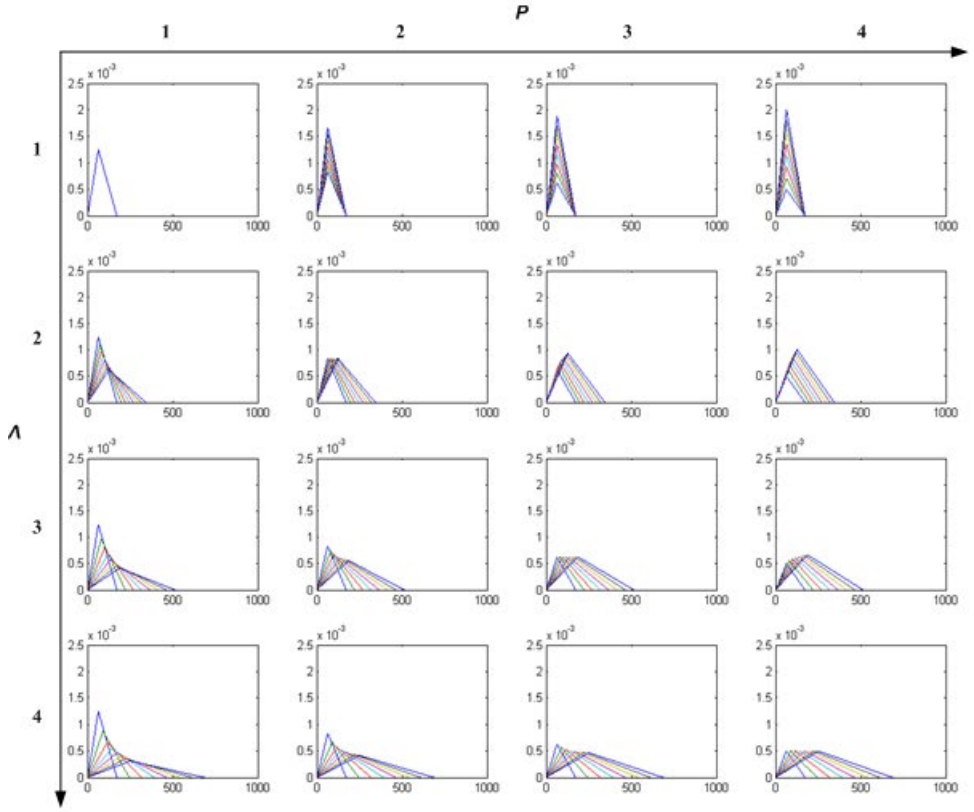


(a) Current waveforms $i_m(t)$

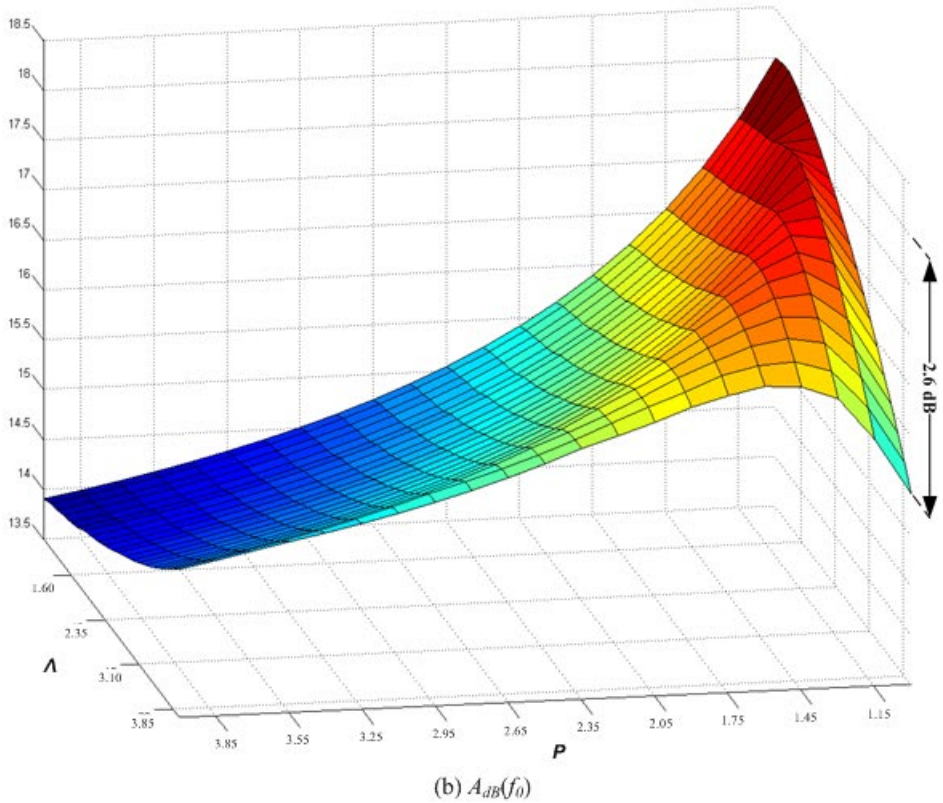


(b) $A_{dB}(f_0)$

Figure 6.7 Spectral peak attenuation at clock frequency versus Λ and P: $M = 4$



(a) Current waveforms $i_m(t)$



(b) $A_{dB}(f_0)$

Figure 6.8 Spectral peak attenuation at clock frequency versus Λ and P: $M = 8$

6.6 Summary

Plesiochronous design with power-balanced system partitioning has been proposed in this chapter to cope with digital switching noise. Efficiency and generality account for its significance of practical applications. With M clock domains, a reduction of $20logM$ can be achieved on the current spectral peaks at the low-frequency clock harmonics. Remarkable noise attenuation thus can be achieved by fine-grained GALS partitioning. It is applicable with marginal performance loss and hardware cost in most cases. Also, dual-clock FIFO and pausible clocking both can be employed for plesiochronous clocking. In Chapter 8, we will further detail the practical implementation of low-noise GALS systems with performance and hardware efficiency.

Also presented in this chapter is a systematic spectrum analysis of switching current with regard to the supply current shaping and spread spectrum clocking techniques. Both solutions are found to be powerful, especially in reducing the switching noise at the high frequency harmonics. Note that, in the framework of power-balanced plesiochronous design, clock phase and frequency modulations can be further applied on each locally synchronous block. The combined scheme will take advantage of each approach. We leave this topic for future investigation.

Chapter 7

GALAXY – A GALS Design Flow

Implementing a GALS system is challenging, because most commercial CAD tools are tightly optimized for synchronous design. This chapter outlines a GALS design flow, named *GALAXY*, which has been successfully applied to our work. In two aspects it can be characterized. First, it is developed especially for the pausible clocking based GALS integration. The challenges imposed by asynchronous-synchronous mixed-time design are addressed at the system level. Second, it is specified with an emphasis on the hard-ware efficiency of GALS implementations. Power and area consumptions are taken into account for the system partitioning in an early design stage. Additionally, an overview on testability issues of GALS systems is presented.

7.1 GALS Design Flow

Figure 7.1 illustrates the *GALAXY* design flow, which is adapted especially for the GALS design based on pausible clocking. For clarity, only the most important steps are depicted and all the iteration paths are omitted. The CAD tools, adopted for each step in our work, are also annotated for reference. As can be seen, hierarchical design has been employed, which essentially involves two layers. On the bottom level are the designs of synchronous functional modules (shown in the left column) and the asynchronous inter-face modules (the right column). These two design stages are independent of each other and can be performed in parallel. On the top level is the integration of a GALS system (the middle column), which accounts for the assembly and verifications of all the pre-developed modules at the system level. Note that the hierarchical design solution is of crucial importance for us, because it allows for the efficient applications of mainstream CAD toolsets to GALS design. We expect this will offer the industrial designer access to the potential of GALS methodologies.

7.1.1 Design of Synchronous Functional Modules

Without loss of generality, we assume that a synchronous system design is already available, which has the functionality validated. Its behavior-level description, typically specified in VHDL or in Verilog, lays the foundation of our work. In this case, adapting a synchronous system to be globally asynchronous necessitates the latency-insensitive conversion of the design. That is, the system should be able to tolerate, to certain extent, the latency uncertainty on communications across clock domains. We address this issue in two steps, as discussed below.

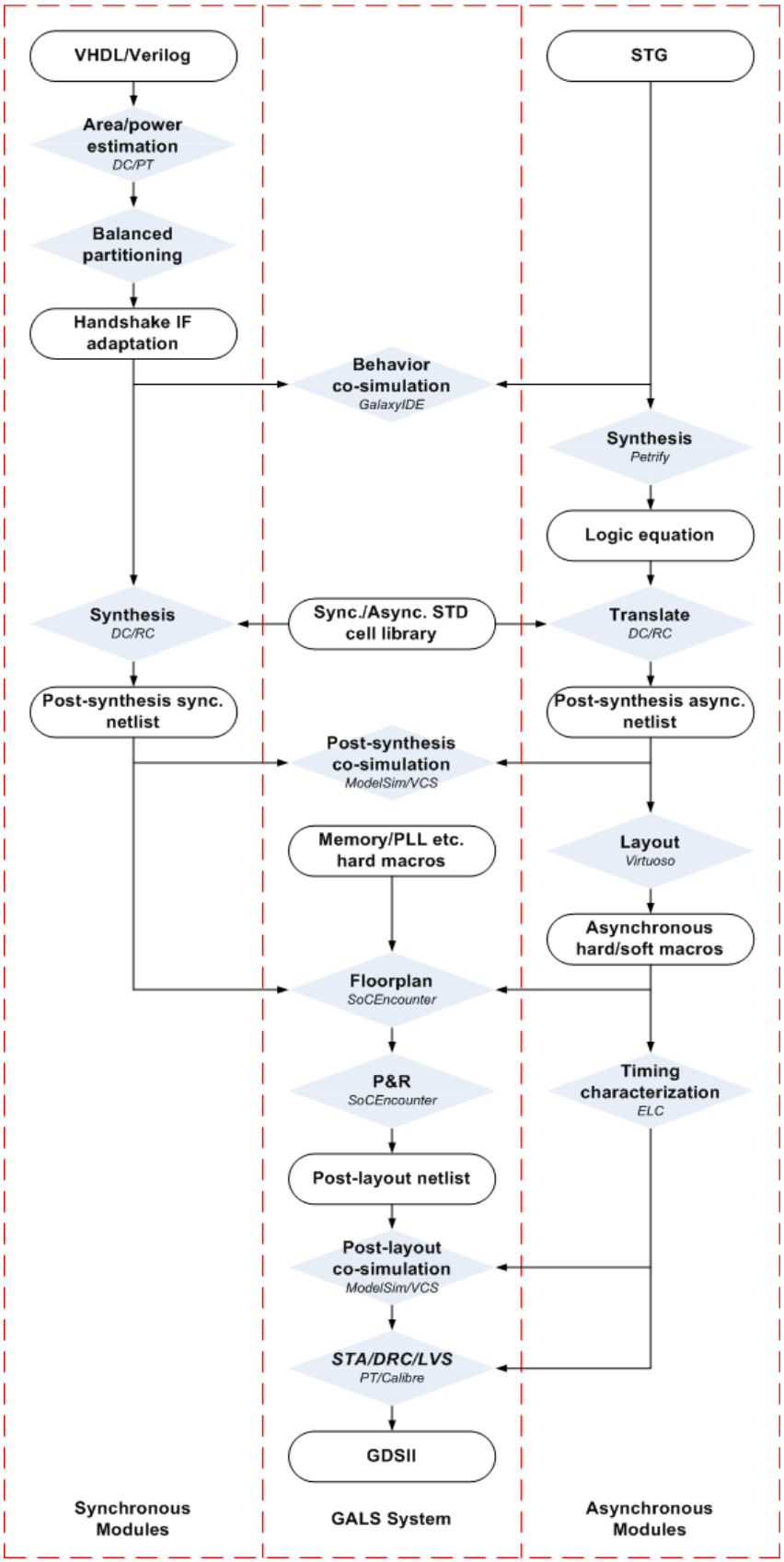


Figure 7.1 GALS design flow – *GALAXY*

System partitioning accounts for our first step towards latency-insensitive design. It determines the location where delay uncertainty could be incurred due to asynchronous communications. Traditionally, GALS partitioning is derived according to the dataflow: group the functional modules with tight data dependency into a common clock domain, and divide the functional modules with loose data coupling into different clock domains. This avoids frequent arbitrations for clock domain crossing. Our work also accounts the physical criteria in addition to the dataflow. All GALS blocks are further balanced with regard to power and area. As addressed in Chapter 6, power-consumption balancing is preferred for us, if the switching noise reduction is of importance. In terms of the area-balanced GALS partitioning, it has been explored in a few studies for the power saving of clock buffer trees [74]. Based on the synchronous design, preliminary synthesis can be performed on the entire system. Then it is straightforward to estimate the power and area of each functional module using commercial CAD tools. In our work, for example, Synopsys *PrimeTime* (*PT*) and *DesignCompiler* (*DC*) were applied, as shown in Figure 7.1, for the power and area estimations, respectively.

Latency insensitivity essentially necessitates distributed flow control on datapath. Given a GALS partitioning scheme, the synchronous design has to be further adapted to allow for the handshake communications (*req*/*ack* or *valid*/*done* for instance) crossing clock domains. Although it is always achievable to accommodate handshaking in a synchronous circuit, the design effort, however, varies remarkably according to the design styles. For a synchronous design with centralized and/or rigid cycle-based flow control, it actually could be challenging. On the other hand, the granularity of partitioning also matters for this issue. Given a medium-to-coarse GALS partitioning, marginal design effort is introduced typically. This will eventually lead to a group of latency-insensitive GALS functional blocks, each triggered by a local clock signal. The example designs of handshake-adapted synchronous FSMs, which are specified for the pausible clocking scheme, can be found in Chapter 5.

7.1.2 Design of Asynchronous Interface Modules

For the pausible clocking scheme, asynchronous FSM design is applied on the I/O port controllers. As seen in Chapter 4, the behavior of a port controller can be modeled by the signal transition graph (STG). Then *Petrify* is used in our work to derive its logic equation [38]. Mapping the logic equation to the target cell library, the gate-level netlist of a port controller can be obtained. The *Petrify* synthesized circuit is hazard free, however, under the timing condition of isochronous forks. The localization of interconnects for each port controller is therefore necessary by the implementation of hard (transistor level) and/or soft (gate level) macros. Due to the simple structure (see Figure 4.8), the interconnect delays in an asynchronous macro are normally negligible, guaranteeing the functional correctness. Timing characterization needs to be carried out on each macro used for the system-level integration. Cadence *Encounter Library Characterizer* (*ELC*) can be employed for this purpose.

The last two decades witness the dramatic progress in the synthesis of asynchronous circuits. Two representative academic tools have been well developed: *Petrify* [38] and *Minimalist* [75], in terms of the asynchronous FSMs design. Rather than *Petrify*, which is based on the STG specifications in the input-output mode, *Minimalist* uses the burst-mode (an extended fundamental mode allowing multi-input changes) specification for hazard-free synthesis. A *Minimalist* derived FSM works under the timing constraints in two aspects. First, the inputs cannot change until the circuit has stabilized: fundamental mode assumption. Second, the feedback paths cannot overtake the forward paths from the same input. When applying to the I/O port controller design in the pausible clocking scheme, both constraints are easily satisfied and verified, as addressed by [76].

7.1.3 Integration of GALS System

Despite the small circuit, an asynchronous FSM can exhibit complicated behaviors. Its state depends not only on the input values, but also on the input sequences. Illegal transitions might drive the circuit into undefined states and cause output glitches. This has been known as the most error-prone issue for pausible clocking based GALS design. To cover all corner cases, the synchronous modules and the asynchronous ports need to be integrated in a complete environment for system simulation. An integrated development tool, named *GalaxyIDE*, has been introduced in [77] for this purpose. It allows the co-simulations of SystemC, VHDL, Verilog, Petri-net and STG. Based on *GalaxyIDE*, we can thus evaluate the functionality and the performance of an entire system in a very early design stage. This is of importance for the large-scale GALS design.

For the back-end design, the structural layout has been used in our work. The pre-developed asynchronous macros are integrated at the top level, like other hard IP cores, such as memory cells and PLLs. All the functional modules, divided into several clock domains, are placed and routed as a multi-clock synchronous design, which can be well supported by standard layout tools. As an example, Figure 7.2 shows the floorplan view of a GALS FFT processor based on pausible clocking.

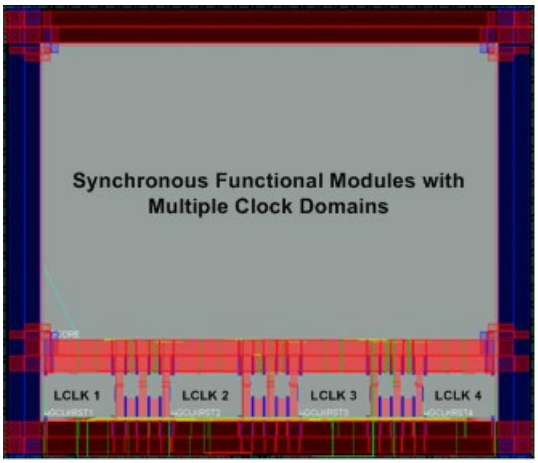


Figure 7.2 Floorplan view of a GALS FFT processor – an example

7.2 Testability of GALS Design

GALS design relies on asynchronous circuits for communicating across clock domains. This, however, challenges the traditional test methodology of digital ICs. Testability of GALS systems is a topic drawing great interest of research. A comprehensive overview on this issue is beyond the scope of this chapter. Below we would like to brief the major difficulties and the representative techniques. A recent survey related to the design for test (DfT) of GALS can be found in [78].

Testing of asynchronous interface circuits is a critical issue [79]. This is especially the case for the pausible clocking scheme where fully asynchronous FSMs are adopted for I/O port design. To avoid hazard on outputs, redundant gates are often introduced, which may cause fault masking in the test of port controllers. The detections of stuck-at faults and delay faults both can be impacted, consequently sacrificing the test coverage. However, considering the limited complexity, asynchronous port controllers can be well verified by functional tests. This was done in [80] by decoupling the I/O ports from the synchronous cores in the test mode, and carrying out data transfer over each of the data links, i.e., a pair of I/O ports, under the control of an extended test wrapper.

Second, asynchronous communications lead to indeterminate operations of a GALS system. Arbitration between asynchronous inputs gives rise to timing uncertainty. Valid data are indicated by handshake signals, rather than by clock cycle. Unfortunately, non-determinism complicates the functional test of a GALS design using the automatic test equipment (ATE). Today most ATEs are strictly cycle based, i.e., only supporting the deterministic timing. To handle this issue, a design style, named synchro-token, was reported in [81]. The authors suggested eliminating non-determinism in GALS design by avoiding arbitration, which seems, however, feasible only for particular applications. A general solution of functional test is to access GALS systems via mixed-time FIFOs on board. On the ATE side, the FIFOs are synchronized by the test clock. Whenever the input FIFO is sufficiently empty, the ATE will write a block of test vectors into it. Once the output FIFO is to be full, the ATE will read a block of test results out of it. Such a coarse-grained flow control is practically achievable for nowadays ATEs [78]. On the GALS DUT side, each FIFO can be equipped with an asynchronous handshake interface to support latency-insensitive communications.

Instead of using ATEs, functional test of GALS systems can be performed based on built-in self-test (BIST). Typically, it is carried out by generating test pattern using linear-feedback shift registers (LFSRs), and compressing the test response using multi-input signature registers (MISRs). Little hardware overhead is introduced as a result. In addition, it supports at-speed testing, which is of particular significance for GALS design. However, only pass/fail information is produced by the BIST logic usually. This limits the capability of functional debugging. GALS DfT by BIST has been applied to our work, as will be presented in Chapter 8.

Finally, for the structural test, it is straightforward to perform scan test in a GALS system at the block level. That is, for each locally synchronous block, a scan chain can be inserted after synthesis, and an external clock generated by an ATE can be used to trigger the block in the scan mode. Automatic test pattern generation (ATPG) is thus applicable to gain appropriate fault coverage. All the asynchronous ports, however, are bypassed in scan test. Actually, experiments in [82] have shown that only a rather small portion of stuck-at faults (less than 0.2%) are within the asynchronous FSMs [82].

7.3 Summary

This chapter outlines our design flow for the pausible clocking based GALS design, named *GALAXY*. It is grounded on the hierarchical design and layout to facilitate the applications of standard CAD tools. *GalaxyIDE*, an integrated design environment, is introduced to cover the asynchronous-synchronous mixed simulations at the behavior level. The hardware efficiency of GALS design is in particular highlighted by exploring the power and area balanced system partitioning.

GALS adaptation of a synchronous system is a design issue which often gets over-looked. The main task is to introduce distributed flow control in the system to tolerate latency uncertainty on cross-clock-domain communications. In our work it was carried out manually case by case, based on RTL-code analysis and exhaustive behavior-level simulations. Highly depending on the system complexity and the designer experiences, this approach can cause prohibitive design effort, and is quite error-prone. Latency in-sensitive design was first suggested explicitly in [83], where the authors focused their studies within the synchronous design realm. Essentially, it accounts for a prerequisite to implement any asynchronous or GALS system. A sophisticated design methodology, named *Elastic Circuits*, has been developed in [84] to cope with the desynchronization of a system at the fine-grained (pipeline stage) level of granularity. Until now, however, there remains a gap to be filled on the GALS adaptation of a synchronous design at the coarse level of functional modules.

Chapter 8

Applications

We have applied the GALS design methodology, developed in this thesis, to chip implementations. In the following, three aspects of our work are highlighted. First, it is of obvious significance for us to prove on silicon the feasibility of our design solution, preferably on industry-relevant systems using advanced technologies. Also, we would like to draw a comprehensive evaluation on the practical impacts of our GALS scheme, taking performance, power and area into account. Further, special effort is dedicated to measure and assess the switching noise reduction, coming from the desynchronization by GALS clocking, on the mixed-signal design in particular.

Among a series of four successful applications in the past five years, we are going to address two representative signal processing designs in this chapter. First, a mixed-signal FMCW (Frequency Modulated Continuous Wave) RADAR sensor chip, named *Lighthouse*, will be reported, which was developed especially for automotive electronics and implemented using the IHP 130-nm SiGe BiCMOS technology. Second, an OFDM (Orthogonal Frequency Division Multiplexing) baseband transmitter chip, named *Moon-rake*, will be presented, which was adapted for WLAN (Wireless Local Area Network) communications and fabricated using the TSMC 40-nm CMOS technology. Each chip actually integrated two digital functional cores in parallel on the die: a synchronous de-sign and its GALS counterpart based on pausible clocking. This paves our way towards an objective comparison in a homogeneous experiment setting – identical in function as well as in process – between two design methodologies.

Depending on the functionality and the complexity, each test chip thus presents a unique platform for us to investigate in distinct design aspects. As most of the mixed-signal SoCs, the *Lighthouse* chip necessitated special design effort to cope with digital switching noise. The power-consumption balanced GALS design was performed on the FMCW-RADAR baseband processor for this reason. Its effects were further evaluated by on-chip measurements at the noise-sensitive analog front-end outputs. In contrast, the *Moonrake* chip addressed the typical case of large-scale digital system design. To offer Gigabit throughput, the OFDM transmitter was deeply parallelized and pipelined on the datapath. Performance, power and area efficiency, therefore, account for the challenges regarding its GALS adaptation. A GALS partitioning scheme, balanced in terms of both power and area, was explored for this purpose. We expect that the experimental results derived in our work will help attracting more interest from the industry in GALS design based on pausible clocking.

8.1 *Lighthouse* Chip – A Mixed-Signal FMCW-RADAR Sensor

8.1.1 General Description of FMCW RADAR

FMCW RADAR is widely used for range and velocity detection of objectives. The frequency of a continuous carrier is modulated by a periodic signal, often denoted by v_{tune}, in the transmitter. The reflected wave is detected by the receiver with a round-trip propagation delay. Its frequency deviation from the transmitted wave, referred to as f_{beat}, thus depends on the distance between the RADAR and the reflecting object, and is also impacted by its relative velocity due to Doppler frequency shift. In terms of the linear frequency modulation, which is the typical case in practice, the distance and the velocity are both linearly correlated to f_{beat}. f_{beat} often can be derived by Fast Fourier Transform (FFT). To achieve a high frequency resolution, large-size FFTs are required. Please refer to [85] for details of FMCW RADAR techniques.

8.1.2 Hierarchical View of *Lighthouse* Chip Design

Figure 8.1 illustrates the simplified block diagrams of the *Lighthouse* chip, viewed in a hierarchical perspective. First, sketched in Figure 8.1 (a), is the top-level schematic, with a focus on the RF front-end circuits (yellow shadowed). The harmonic transceiver design is used at the millimeter-wave (MMW) frequency of 120 GHz. A ramp on v_{tune} is generated to control the VCO, resulting in a linear modulation of the carrier frequency. f_{beat} is derived by a baseband processor on the outputs of the sub-harmonic mixer after digitalization. Thanks to the IHP state-of-the-art 130-nm SiGe Bi-CMOS technology, the analog front-end (in SiGe Bipolar) and the digital baseband (in CMOS) have been fully integrated on chip. It offers a low-cost and compact design solution to the FMCW-RADAR sensors, which is in demand by automotive applications. The design details of the *Lighthouse* MMW transceiver can be found in [86].

Our work concentrated on the design of the baseband processor. Its data flow can be abstracted as Figure 8.1 (b). The design is dominated by two in-parallel 4096-point FFT coprocessors, a synchronous one and its GALS counterpart, to allow a comparison of apple-to-apple. Functional tests of the FFT cores are supported by the on-chip BIST logic, the PRNG and the MISR, as explicitly shown in the figure. To save silicon area, all the data pads are shared by the two FFT cores. A JTAG controller is employed for the configuration of working modes and GALS clocks. Additionally, there is a micro-controller integrated to provide necessary programmability.

The structure of synchronous 4096-point FFT coprocessor is outlined in Figure 8.1 (c). The radix-4 DIF (Decimate-In-Frequency) SDF (Single-path Delay Feedback) pipe-lined design is applied [87]. For the 4096-point FFT, six butterfly stages are introduced, with the signal flow graph as plotted in the shadowed box. Before FFT a pre-processing unit is deployed to insert the Hamming window on the raw data for anti-aliasing. Based on the FFT results, f_{beat} and target range are derived by post-processing.

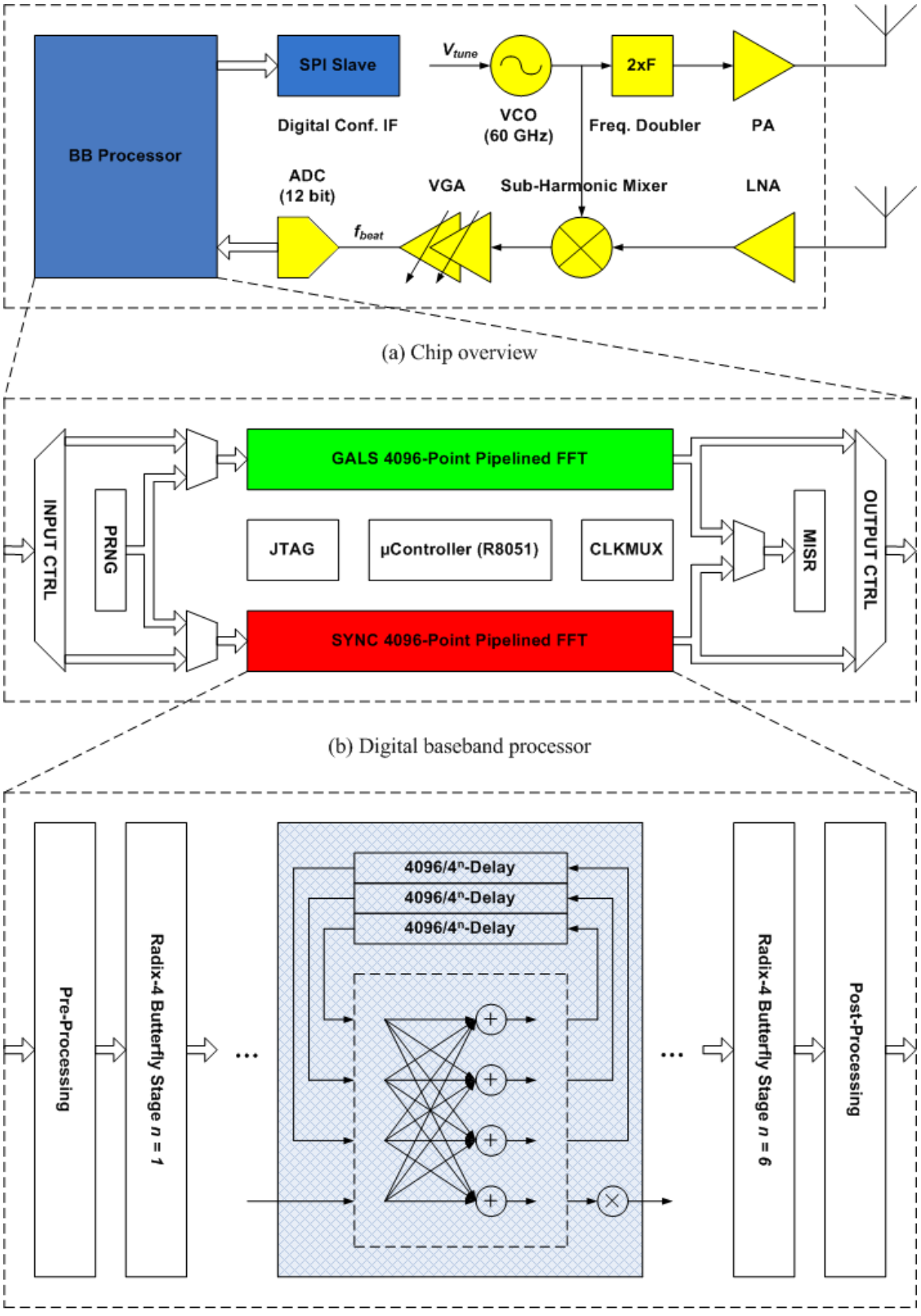


Figure 8.1 Simplified block diagrams of *Lighthouse* chip

8.1.3 GALS FFT Coprocessor

On-chip mixed-signal design exacerbates the substrate noise suffered by the analog circuits dramatically. In case of the *Lighthouse* chip, attention has been paid especially to avoid degrading the phase noise performance of the VCO, due to the switching noise coupled from the digital baseband processor via the common substrate. The phase noise affects the detection accuracy of f_{beat}, even leading to the masking out of weak signals, and is therefore considered a critical factor of the FMCW RADAR sensors.

To address this issue, a GALS FFT coprocessor has been developed for evaluation. Its architecture is shown in Figure 8.2. Five independent clock domains are introduced, based on the pausible clocking scheme. Table VIII.I presents the power and area break-downs of the GALS FFT design, in accordance with the post-synthesis netlists. As can be seen, the power consumption is well balanced over the clock domains. We can thus expect an efficient attenuation of switching noise in comparison with the synchronous FFT core. The area occupied by each GALS block is, however, significantly different. Between the clock domains, the loosely-coupled point-to-point data links, as presented in Chapter 5, are employed for the asynchronous communications. Also highlighted in the figure is the FFT datapath, which consists of five GALS channels. In addition, there are four pairs of data links used in the design. They are especially for the FFT control signals, and only get activated occasionally.

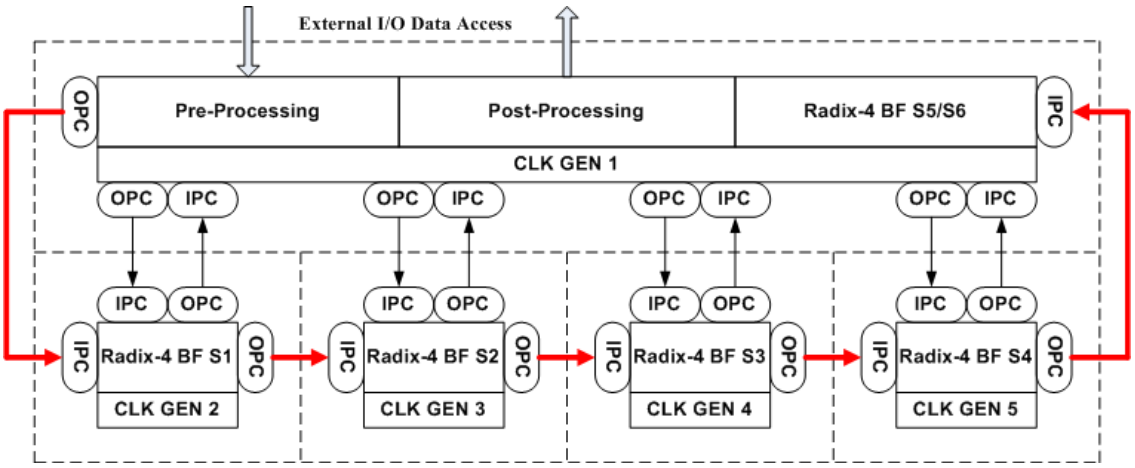
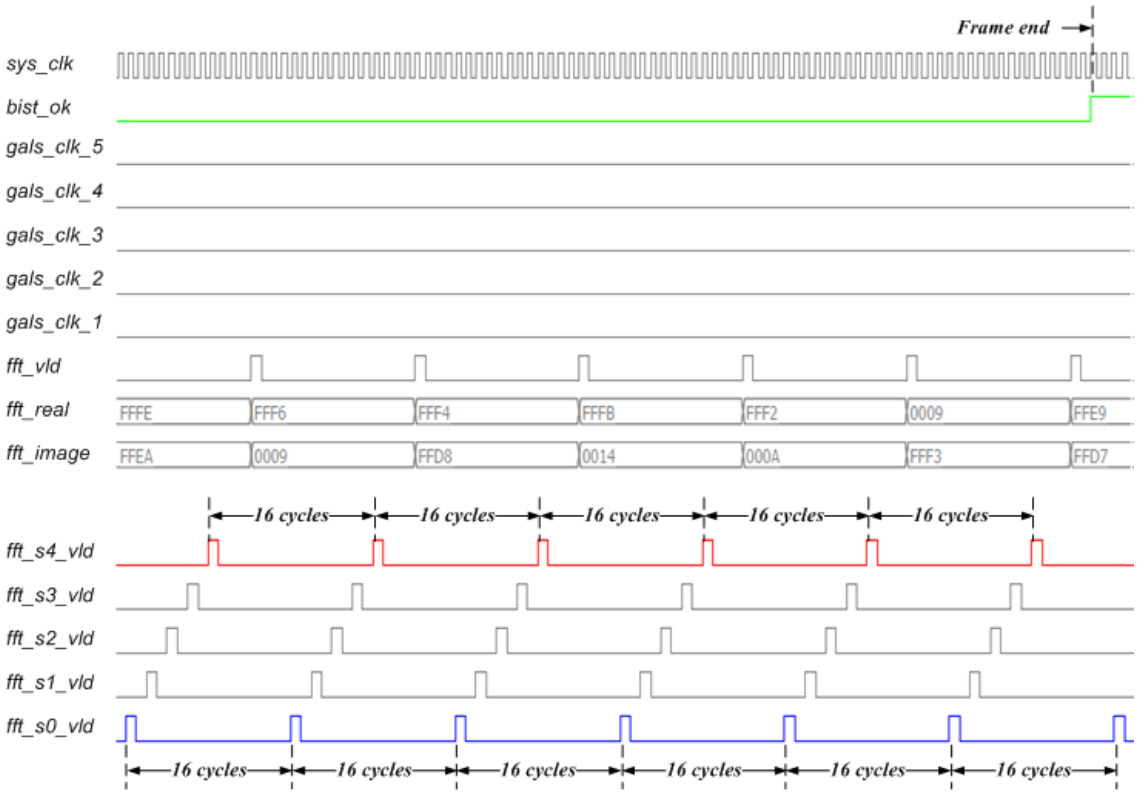


Figure 8.2 Architecture of the GALS FFT design

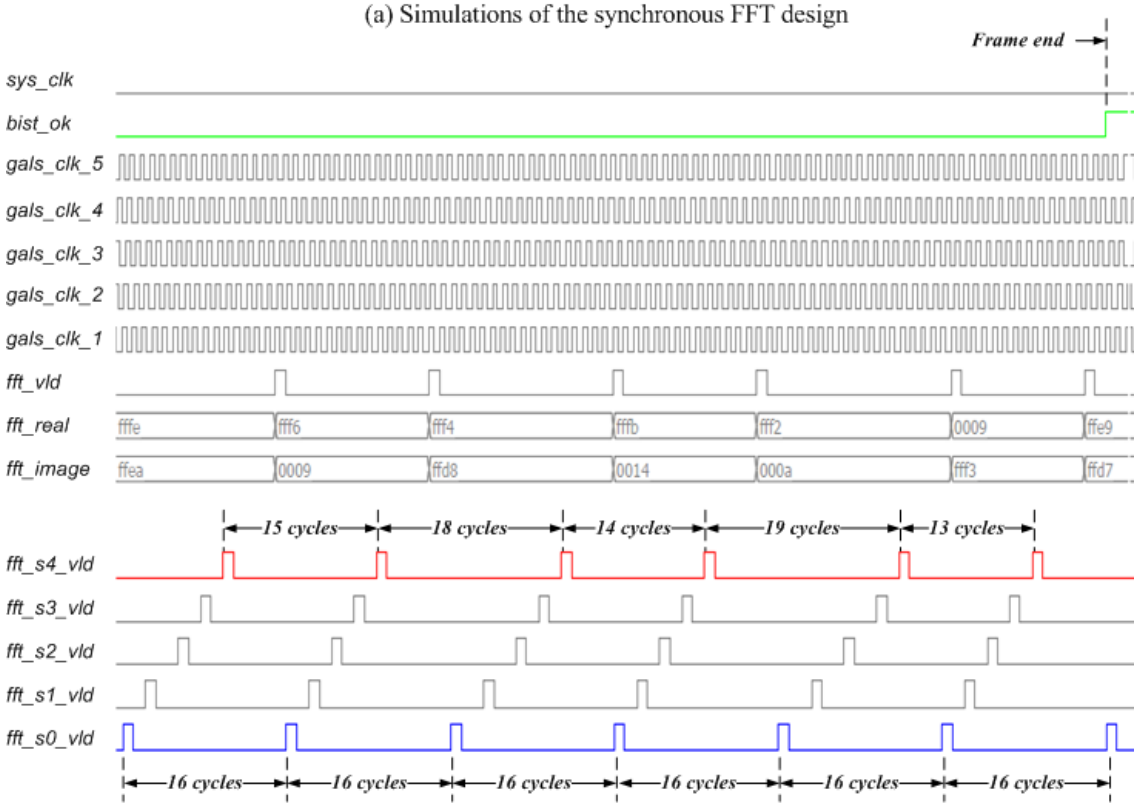
Table VIII.1 Power and area breakdowns of GALS FFT design

	BLK 1	BLK 2	BLK 3	BLK 4	BLK 5	TOTAL
Power (1) (mW)	9.1	9.4	7.7	7.6	8.6	42.4
	21.4%	22.2%	18.2%	17.9%	20.3%	100%
Area (2) (mm^2)	0.57	0.56	0.27	0.21	0.19	1.80
	31.7%	31.1%	15.0%	11.7%	10.6%	100%

(1) The power was estimated by *Synopsys PrimeTime* based on the post-synthesis simulations at 50 MHz.
(2) The area was estimated by *Synopsys DesignCompiler* based on the post-synthesis netlists.



(a) Simulations of the synchronous FFT design



(b) Simulations of the GALS FFT design

Figure 8.3 Timing uncertainties on the GALS FFT datapath – an example

Latency uncertainty can be incurred on the FFT datapath due to the asynchronous communications. As an example, Figure 8.3 compares the simulation waveforms when the synchronous and the GALS FFT cores are enabled in the BIST mode, respectively. A sample to the FFT stages is produced by the PRNG every sixteen clock cycles. Each time a frame of 4096 samples is processed correctly, the MISR will enable the *bist_ok* signal to switch. For the synchronous design, the interval between two successive data is sixteen-cycle throughout the datapath. In terms of the GALS FFT design, however, extra cycles could be required for input synchronization at the clock boundaries. This gives rise to a remarkable timing uncertainty on the GALS datapath, as manifested by Figure 8.3 (b). Handshaking ensures robust communication in this scenario.

8.1.4 Chip Implementation

The hierarchical design, as outlined in Chapter 7, was employed on the GALS FFT core. Appendix B details the timing constraints of synthesis for reference. The final lay-out of *Lighthouse* chip is presented in Figure 8.4. The power supplies are separated between the digital baseband processor and the MMW transceiver. A probe pad, which is connected with the digital core-VDD power ring, is placed on the chip for the switching noise measurement. The VCO output, after frequency division, also can be observed to figure out the phase noise performance.

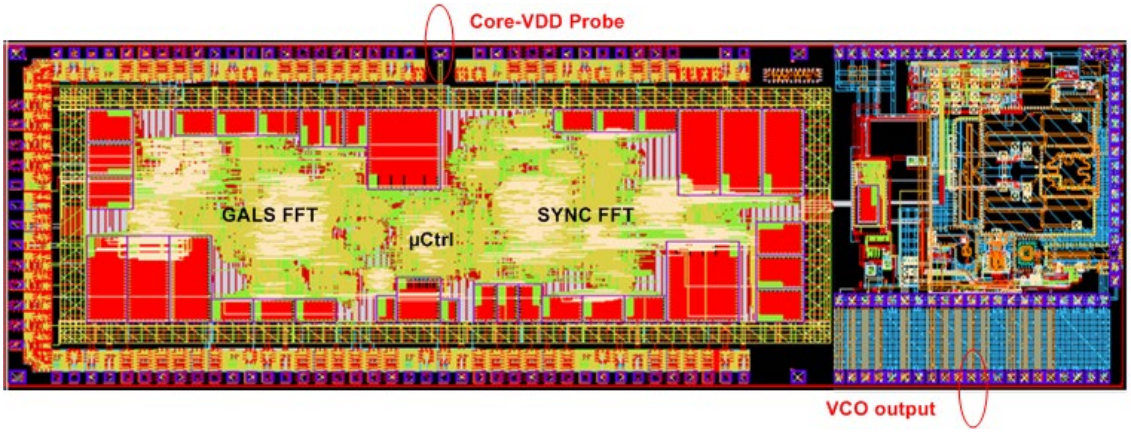


Figure 8.4 Layout view of *Lighthouse* chip

The die size of *Lighthouse* chip is 17.1 mm^2. Over two-thirds of the silicon area is occupied by the digital baseband processor. In total 149 signal and power pads are used. Figure 8.5 shows a photo of the chip, with a 208-pin ceramic QFP package.

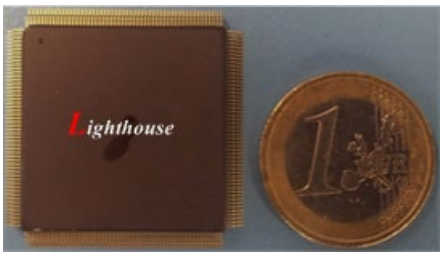


Figure 8.5 *Lighthouse* chip in package

8.1.5 Measurement Results

The *Lighthouse* chip has been tested on the IHP in-house testing platform as seen in Figure 8.6. Both, the on-wafer test and the packaged chip test, were performed using the *Verigy 93000 system*. To facilitate the package test, an adapter board was developed. It is mounted on the system baseboard, and equipped with a specific socket holding the packaged chip during test. Some additional SMA connectors are also provided on the adapter board to connect a spectrum analyzer for the measurements.

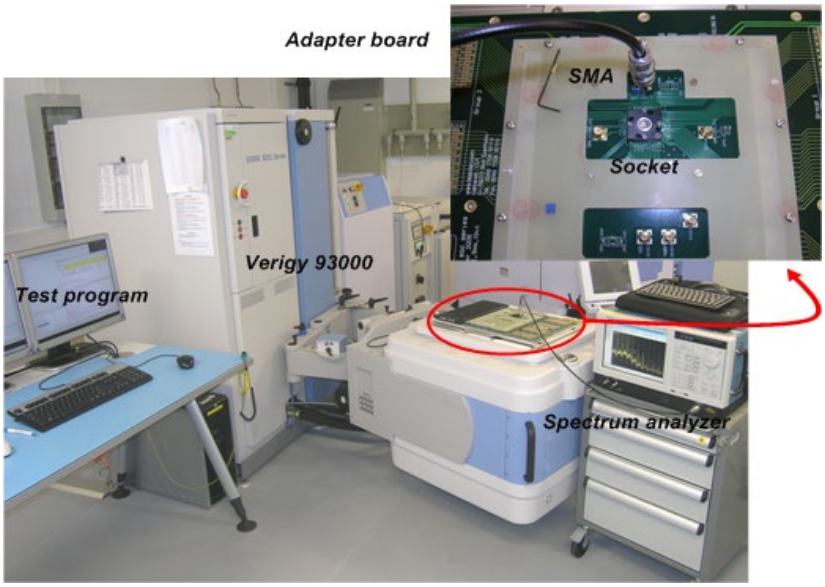


Figure 8.6 IHP in-house testing platform

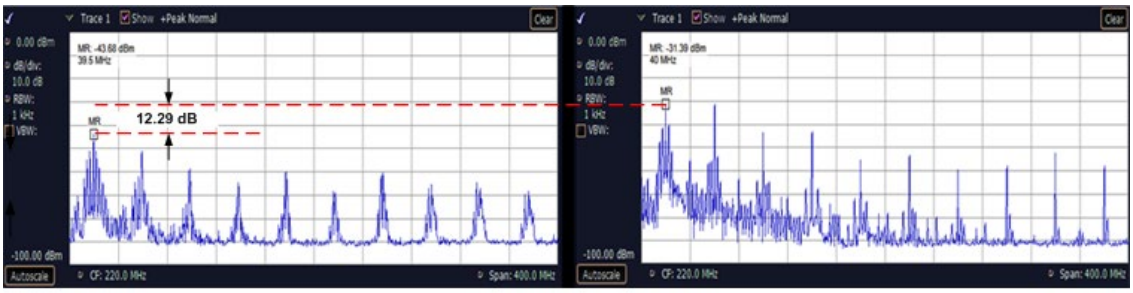
The pipeline design of the GALS FFT coprocessor necessitates the plesiochronous clocking to achieve full performance. In our experiments, all the five local clocks were set to work around a central frequency of 40 MHz, only with slight frequency mismatch. Table VIII.2 is a comparison of major design parameters between the GALS core and the synchronous counterpart. On the GALS design, marginal overheads were measured in power and area, indicating the hardware efficiency of our solution. Data throughput is almost the same. Nevertheless, it is worth to note that no intensive communication is incurred actually: data are transferred discontinuously in the design.

Figure 8.7 illustrates the digital core-VDD spectra, at the first ten clock harmonics, measured when the GALS and the synchronous FFT cores were activated, respectively. With a resolution bandwidth (RBW) of 1 kHz, a peak attenuation of 12.29 dB has been gained by the GALS design at the fundamental frequency. It matches well our analytical results in Chapter 6. Also, we get a spread bandwidth at each harmonic frequency in the GALS design. Figure 8.8 displays in detail by zooming in at the fundamental frequency. Five spectral peaks of the GALS clocking are clearly shown, as characterized in Table VIII.3. A RBW of 10 kHz was used there for the stable measurement. The wide RBW on the other side tends to merge the neighbor spectral components, causing some under-estimation of the measured peak attenuation.

Table VIII.2 Comparisons in power, area and throughput

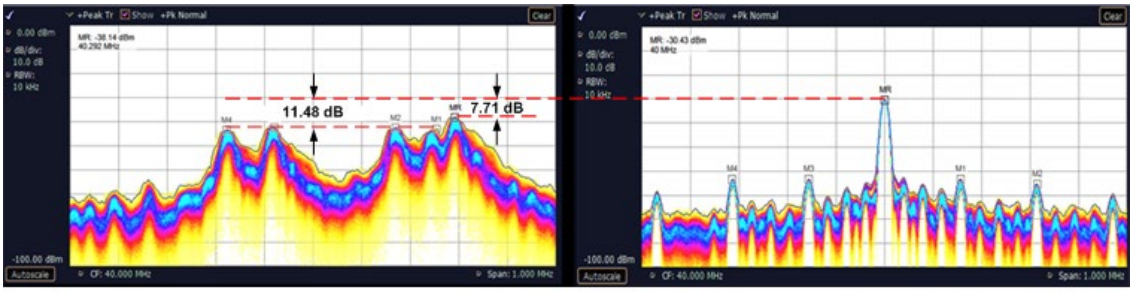
	Power (1)	**Area** (2)	**Cycle num. per frame** (3)
SYNC FFT	29.028 mW	1.84 mm^2	65536
GALS FFT	31.032 mW	1.92 mm^2	66791 on average
Overhead	***6.61%***	***4.35%***	***1.91%***

(1) The power consumption was measured on chip working at 44 MHz;
(2) The area was reported after final layout by Cadence SOCEncounter;
(3) The cycle number was measured on the chip working in the BIST mode.



(a) GALS working mode (b) Synchronous working mode

Figure 8.7 Spectral peaks of on-chip core-VDD at the first ten harmonics (*RBW = 1 kHz*)

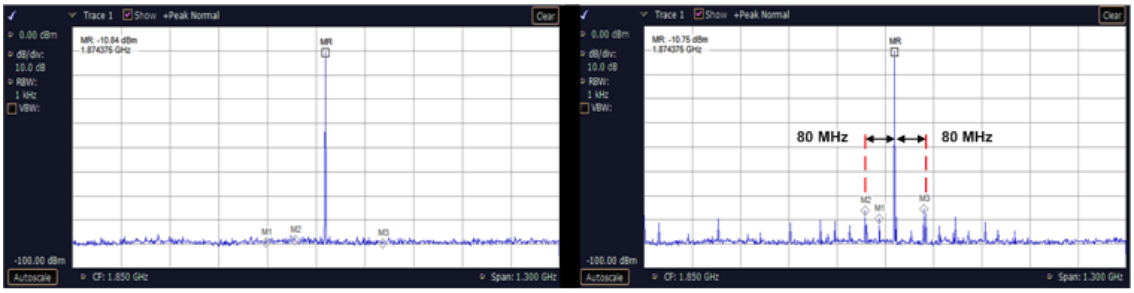


(a) GALS working mode (b) Synchronous working mode

Figure 8.8 Spectral peaks of on-chip core-VDD at the clock frequencies (*RBW = 10 kHz*)

Table VIII.3 Spectral peak characterization of the GALS clocks

	MR	**M1**	**M2**	**M3**	**M4**
Frequency (MHz)	40.29	40.26	40.17	39.92	39.83
Spectral Peak (dBm)	-38.14	-43.15	-42.57	-42.41	-43.44



(a) GALS working mode (b) Synchronous working mode

Figure 8.9 Spectrum of VCO output signal (*RBW = 1 kHz*)

Finally, the phase noise measured on the VCO output signal, after a 1/32 frequency division, is presented in Figure 8.9. Spectral spurs were detected, when the synchronous FFT core was activated. The most outstanding noise peaks have been found to be +/- 80 MHz apart from the VCO oscillating frequency, with 15 dB above the continuous noise floor. Obviously, they are introduced by the on-chip digital switching noise propagating through the common substrate, from the baseband processor to the MMW transceiver. When the GALS FFT coprocessor was working instead, there was no evident spectral spur measured on the VCO output. This means, with clock desynchronization the power balanced GALS design also leads to a dramatic reduction of substrate noise. However, the relatively long distance of the GALS FFT core, compared with the synchronous one, to the analog front-end circuits should be taken into consideration as well. In general, it accounts for a better physical isolation to prevent the propagations of digital switching noise from the GALS design, and in turn improves the phase noise performance of the VCO, to some degree.

8.2 *Moonrake* Chip – An OFDM Baseband Transmitter

8.2.1 General Description of OFDM Transmission

Recent advances in the wireless communication techniques push new applications. Several international standards, which are being drafted, already support the capability of gigabits per second (Gbps) of data rate. Systems operating in the 60-GHz band draw special attention of research due to the available 7 GHz of unlicensed spectrum. OFDM modulation is often employed, considering its high spectral efficiency in multiple path propagation environments [88].

IHP has been dedicated to the development of advanced wireless communication systems for years. In [89] the implementations of our 60 GHz SiGe-BiCMOS chipsets for the OFDM transmission are reported in detail. The main physical layer parameters are summarized in Table VIII.4. The system is essentially based on the 256-point IFFT processing, with 192 points for data transmission. A symbol period of 800 ns is defined, 20% of which is occupied by the cyclic prefix to tolerate multipath delay spread. Fixed modulation modes are supported, ranging from BPSK to 64-QAM. Using the 16-QAM modulation, the maximum data rate of 960 Mbps can be achieved. To our knowledge, this represents are state-of-the-art of the OFDM links with silicon based circuits in the 60 GHz band.

Table VIII.4 Physical parameters of the OFDM scheme

FFT Bandwidth	**FFT Size**	**Subcarrier Spacing**
400 MHz	256 Point	1.5625 MHz
Data Subcarriers	**Pilot Subcarriers**	**Zero Subcarriers**
192	16	5
Symbol Duration	**FFT Interval**	**Guard Time**
800 ns	640 ns	160 ns

8.2.2 Synchronous Baseband Transmitter

Figure 8.10 depicts the data flow structure of the OFDM baseband transmitter (BB TX). To achieve Gigabit throughput, the BB TX design is highly parallelized and pipelined. The details regarding its synchronous implementations can be found in [90]. Below we would like to outline the general functionality. Data items are first acquired by the input control unit. A FIFO is deployed for data buffering. According to the mode of modulation, the input controller initializes the number of OFDM symbols to be transmitted, and also performs the cyclic redundancy check for the signal field. The source data is scrambled to avoid long strings of zeros or ones. This contributes to limiting the power peak in an OFDM symbol.

Furthermore, FEC (Forward Error Correction) coding is carried out. Similar to the 802.11 a standard, convolutional (171, 133) codes are utilized with a constraint length of seven and a code rate of ½. However, the Giga-bit data rate significantly challenges the implementation of Viterbi decoders, which allows a throughput of only one bit per clock cycle. To address this issue, the data stream is split into a group of sub-streams in the BB TX, each getting encoded separately. Up to twelve code streams are supported, but the exact number depends on the source data length and the modulation mode. Each code stream consists of one OFDM symbol. A form of TDMA (Time Division Multiple Access) is adopted for the frame division among the encoded streams. A middle stage controller is dedicated to the stream arrangement. In the receiver, the sub-streams can be decoded concurrently, thus sustaining the Giga-bit data rate.

For OFDM combined with convolutional codes, interleaving is mandatory to resist the frequency-selective fading. Two permutations are involved basically. First executed is subcarrier interleaving in order to spread adjacent code bits across the spectrum. The successive bits are further mapped to bit positions with alternative reliability. A total of six parallel interleavers are employed in the BB TX design. Consequently, glue logic is required to collect and distribute the coded data. Interleaved streams have to be assigned to the appropriate data subcarriers. Additionally, a random sequence feeds the pilot subcarriers. Before performing the IFFT, the data streams will get mapped in accordance with the modulation mode.

The 256-point IFFT is decomposed into four 64-point IFFTs in parallel, based on the Radix-4 algorithm. The DIT (Decimate-In-Time) SDF pipelined design is applied to each of the 64-piont transforms. Its butterfly structure can be referred to Figure 8.1 (c). The outputs of the 64-point IFFTs are combined by a 4-point IFFT. Therefore, an IFFT throughput of four samples per clock cycle is achieved. Given the sampling rate of 400 MHz, which is actually defined by the FFT bandwidth, a working frequency of no less than 100 MHz is sufficient for the BB TX design. After the 256-point IFFT, cyclic prefix is inserted against ISI (inter-symbol interference) and ICI (inter-carrier interference). For transmission the IFFT data are finally multiplexed with the preamble. To guarantee good synchronization and initial channel estimation, a long preamble of eleven OFDM symbols is used in the design [91].

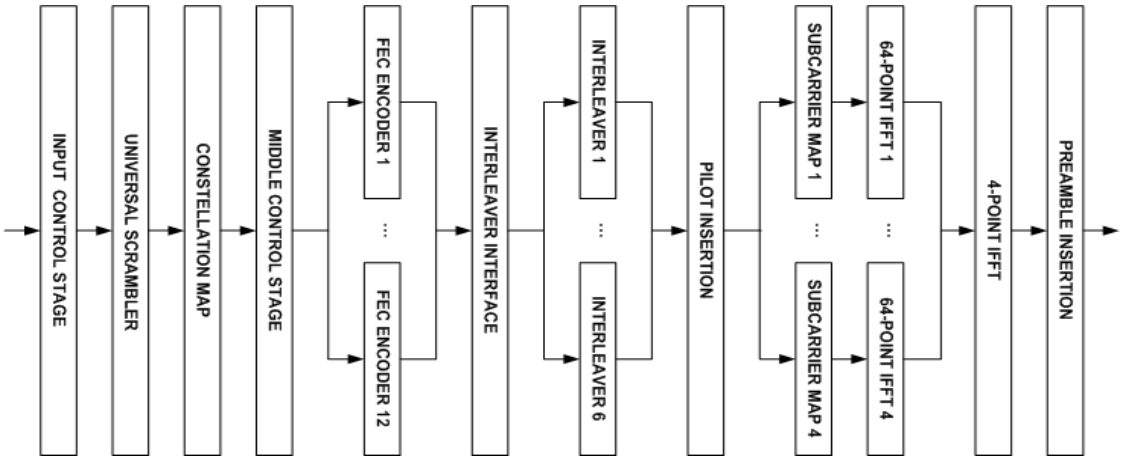


Figure 8.10 Data flow structure of the OFDM BB TX

The synchronous BB TX design has been validated on the Xilinx Virtex II FPGA platform with full functionality. Approx. 18,000 flipflops are used, along with as many as 114 embedded memory blocks. There are additionally over forty real-value multipliers employed, most of which are applied for the 256-point IFFT processing. According to the synthesis and mapping reports of the FPGA, the design results in a complexity of 7.8 M equivalent logic gates in total [90].

8.2.3 GALS BB TX Design

The complicated structure of BB TX design challenges its GALS implementation. Performance and hardware efficiencies account for our major design goals. Figure 8.11 illustrates the architecture of GALS BB TX developed in this work. As can be seen, the entire design has been partitioned into six individual clock domains. All tightly coupled control and data pre-processing modules, except for the interleavers, are grouped into a single GALS block. Considering the large expenses of power and area, the interleaver bank is separated into three clock islands, each having two interleavers in parallel. The most computation-intensive processing of the 256-point IFFT is broken down into two GALS blocks. The four 64-point IFFTs are done in a common clock domain. The post-processing of 4-point IFFT is performed in an independent clock region, together with the preamble insertion. Between the different clock domains, there are sixteen point-to-point GALS data links for asynchronous communications. Half of them (highlighted by the red lines in Figure 8.11) are dedicated to the datapath. As detailed in Chapter 5, the loosely-coupled design has been applied.

Table VIII.5 presents the power and area breakdowns of the GALS BB TX design. We see that all the clock domains are well balanced both in power consumption and in area. Actually each GALS block is at most 11% higher in power and 15% larger in area than the average. Note that area balancing is of particular significance for the hardware efficiency of complex GALS systems. It is beneficial for the interconnect localization and the wire length minimization in the layout, and therefore, contributes to the timing convergence at the system level.

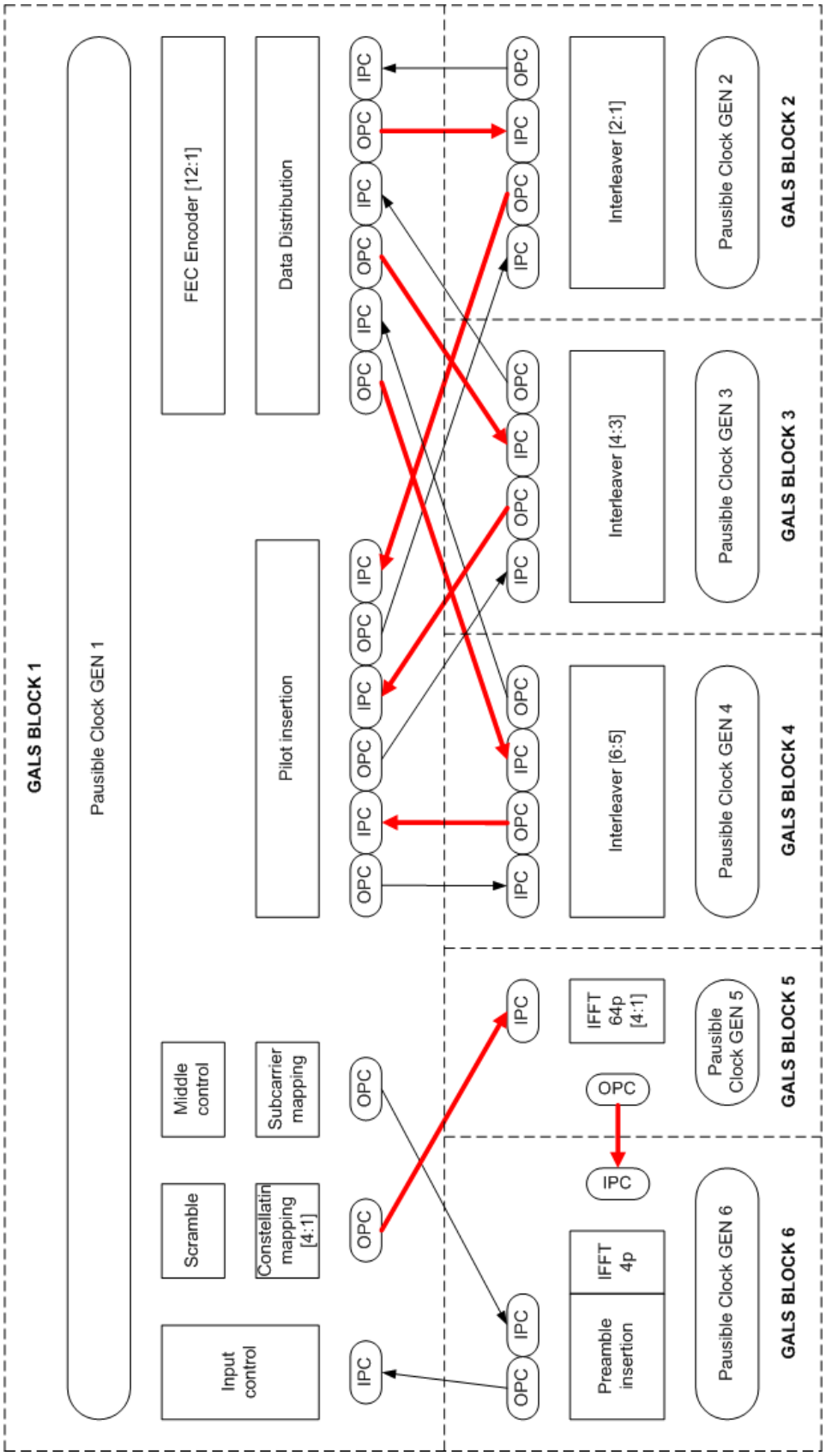


Figure 8.11 Architecture of the GALS OFDM BB TX design

Table VIII.5 Power and area breakdowns of the GALS BB TX design

	BLK 1	BLK 2	BLK 3	BLK 4	BLK 5	BLK 6	TOTAL
Power (1) (mW)	26.1	40.2	40.2	40.2	40.9	42.4	230
	11.5%	17.4%	17.4%	17.4%	17.8%	18.5%	100%
Area (2) (mm^2)	0.42	0.39	0.39	0.39	0.22	0.32	2.2
	19.3%	17.8%	17.8%	17.8%	10.3%	17.0%	100%

(1) The power was estimated by *Synopsys PrimeTime* based on the post-synthesis simulations at 160 MHz.
(2) The area was estimated by *Synopsys DesignCompiler* based on the post-synthesis netlists.

8.2.4 Chip Implementation

The synchronous and the GALS BB TX designs have been integrated in parallel on the same chip, named *Moonrake*. The state-of-the-art TSMC 40-nm CMOS technology is used in the chip implementation. For the GALS design, hierarchical layout was performed. All local clock generators and the asynchronous I/O port controllers were first implemented at the transistor level, and then integrated on the chip as hard macros. The GALS clock frequency can be configured via a JTAG interface. Further, there is a PLL equipped on the chip, which is required for the synchronous BB TX design to generate the global clock signal.

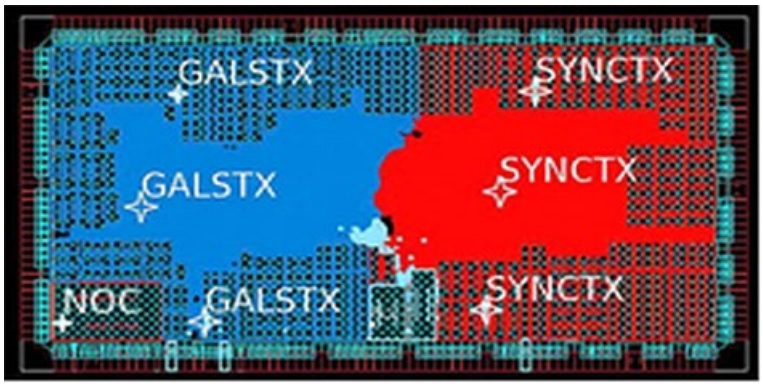


Figure 8.12 Layout view of *Moonrake* chip

The *Moonrake* chip is 9 mm^2 (4000 μm^2 × 2250 μm^2) in die size. Most silicon area is actually occupied by the on-chip memory blocks. A total of 219 I/O and power pads are used. Figure 8.13 demonstrates a photo of the *Moonrake* chip, with a 345-pin BGA package.

Figure 8.13 *Moonrake* chip in package

8.2.5 Experimental Results

A. Area and Power

First addressed are the hardware costs of the GALS infrastructure based on pausible clocking. The post-layout area and power estimations of the local clock generators, the asynchronous port controllers and the ME-latching registers are shown in Table VIII.6. The pausible clocking specific circuits in total account for only 1.6% in area and 2.7% in power dissipation of the entire GALS BB TX design. Most of the expenses result from the on-chip ring oscillators for clock generation.

Table VIII.6 Area and power of GALS Infrastructure

	Local Clock Gen		**Async. Port Ctrl.**		**Input Data Registers**	
Area (μm^2)	30,000	1.25%	640	0.03%	5,700	0.26%
Power (mW)	4.05	1.77%	0.19	0.08%	1.80	0.79%

The clock and reset buffer trees in the synchronous and the GALS BB TX designs after layout are characterized in Table VIII.7. The GALS design gives rise to a reduction in clock tree complexity. A 2.7X drop of the buffer level and a 6% saving of the buffer number are achieved, with a 24% reduction of clock tree power. In addition, the GALS design leads to a 30% drop of the reset tree buffer number.

Table VIII.7 Characterization of clock and reset buffer trees

	Clock Trees				**Reset Trees**	
	Number	Level	Buffer #	Power	Number	Buffer #
SYNC BB TX	1	27	1645	42 mW	1	582
GALS BB TX	6	≤ 10	1549	32 mW	6	405

Table VIII.8 reports the cell areas of two BB TX designs. The GALS design gave a slight overhead (0.57%) after synthesis over the synchronous one (w/o PLL). During the layout, however, we noticed a 1.16% shrink of the area of the GALS core and in contrast a 1.26% increase of the synchronous core. This led to a 0.67% area saving in the GALS design eventually. Similar comparisons are also done in terms of the power dissipation. As shown in Table VIII.9, the GALS BB TX design accounts for a power reduction of 5.8%, measured on the chip, over its synchronous counterpart (w/o PLL). By taking the on-chip PLL ($0.1\ mm^2, < 1.5\ mW$) into account, a more objective comparison reveals a 5% reduction of area and a 6% saving of power obtained by the GALS design.

The above comparisons manifest the benefits of GALS design for layout. The chip-wide cell location and synchronization challenged the timing closure of the synchronous BB TX design. In contrast, the area balanced partitioning engaged in our GALS design resulted in a set of compact locally-timed blocks, which can be optimized in layout more efficiently. As a consequence, the hardware costs of the GALS infrastructure were fully compensated at the system level. It is the first time that solid experimental results have been reported on the area and power efficiency of GALS design.

Table VIII. 8 Comparison in cell area

	Post-synthesis netlist	**Post-layout netlist**	**Difference**
SYNC TX w/o PLL	2,206,895 μm^2	2,234,712 μm^2	+27,817 μm^2
GALS TX	2,225,823 μm^2	2,220,080 μm^2	-25,743 μm^2
Difference	+18,928 μm^2	-14,632 μm^2	

Table VIII. 9 Comparison in power dissipation

	Post-layout simulation	**Chip measurement**	**Difference**
SYNC TX w/o PLL	234 mW	252 mW	+18 mW
GALS TX	225 mW	237 mW	+12 mW
Difference	-9 mW	-15 mW	

B. Data Throughput

The GALS BB TX design provides a perfect platform to evaluate the performance of the pausible clocking scheme. To attain full performance, all the GALS blocks need to working around a central frequency, i.e. in the plesiochronous clocking mode. Each of the data frames is transferred continuously across clock domains. Therefore, the system performance is actually determined by the data throughput of GALS data links, which is susceptible to the handshake loop delays.

A data frame with QPSK modulation was processed for performance evaluation in simulations. The RTL design of GALS BB TX was adapted to support both, the loosely-coupled and the tightly-coupled data links. The synthesized netlists of the asynchronous controllers were used. Given a clock frequency of $160\ MHz$, an acknowledge window of $w_{MUTEX} = 4.4\ ns (0.7T_{clk})$ was reserved. The clock-tree and port-interconnect delays were assigned according to the timing reports after layout (see Table VIII.10).

Implementing the GALS BB TX design using tightly- and loosely-coupled data links was first explored. As shown in Figure 8.14, the system only can offer a throughput of at most one item of data every two cycles by employing the tightly-coupled data links. The loosely-coupled link design, in contrast, boosts the throughput dramatically. Even in the worst (maximum delay) case, the throughput loss is within 9% compared with the synchronous design. Actually, taking the parameters of Table VIII.10 into (4.24), we see that for most data links the throughput-tolerant timing condition was satisfied.

Inserting asynchronous buffers in the GALS data links was further evaluated. The *Mousetrap* buffer design was applied in simulations [52]. As shown in Figure 8.14, with only one stage of buffer deployed on each tightly-coupled data link, we can improve the system throughput over 20%. The loosely-coupled data link design with buffer insertion leads to a negligible performance drop even in the worst case.

Also investigated was the impact of large interconnect delays on GALS data links. It will substantially worsen the system throughput as presented in Figure. 8.15. Design optimization of the port controllers helps little under this condition. However, with only one stage of asynchronous buffer, an interconnect delay of half a clock cycle is tolerated by the loosely-coupled data link with a 5% loss of throughput.

Table VIII.10 Clock-tree and port-interconnect delays of the GALS design

	Clock-tree insertion delay (ps)						Port Int. delay (ps)	
	GALS 1	GALS 2	GALS 3	GALS 4	GALS 5	GALS 6	OPC2IPC	IPC2OPC
Worst case	960	610	670	520	870	810	520	270
Best case	450	380	380	290	430	420	210	140

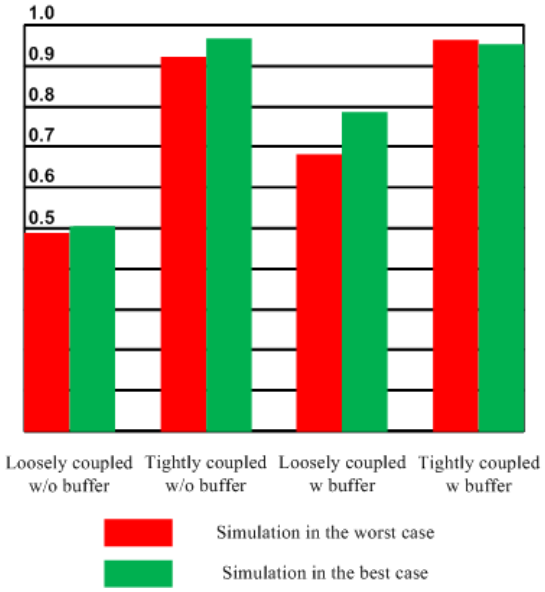


Figure 8.14 Throughput comparison with different GALS data links

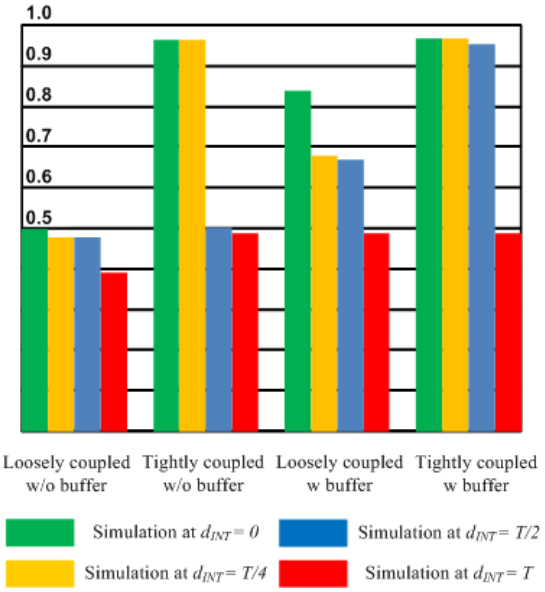


Figure 8.15 Throughput comparison with different interconnect delays

C. Switching Noise

Finally, the digital switching noise reduction using GALS design was evaluated on the chip. Figure 8.16 is the bottom view of the *Moonrake* adapter board. There are two SMA test points provided especially for the switching noise measurements. VDD_AE22 is connected directly to a core-VDD power pad, with no decoupling capacitor in parallel. It thus allows us to monitor the internal supply voltage on chip. The spectral peaks at the first ten clock harmonics, measured when the GALS and the synchronous BB TX cores were activated, respectively, are illustrated in Figure 8.17. An attenuation of over 26 dB has been achieved by the GALS design at the fundamental clock frequency. It is, however, much higher than our theoretical analysis of approx. 16 dB for the power-balanced GALS partitioning with six clock regions. After detailed studies, we found that a simple reason, which is also the best explanation from us, should account for this result. For a large-size chip as *Moonrake*, the supply voltage can be location dependent on the chip. But as seen in Figure 8.16, the measurement pad is at the upper-right corner of the chip, far apart from the GALS design and exactly on the synchronous side. To address a more fair comparison, we additionally measured the board-level power supply at the test point VDD_BOARD. This leads to a spectral peak reduction of less than 20 dB as shown in Figure 8.18, which is close to our analytical estimation.

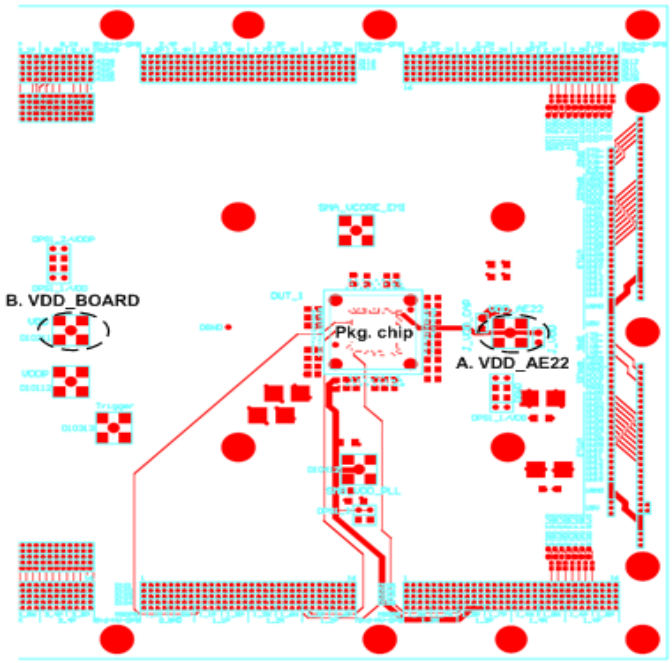
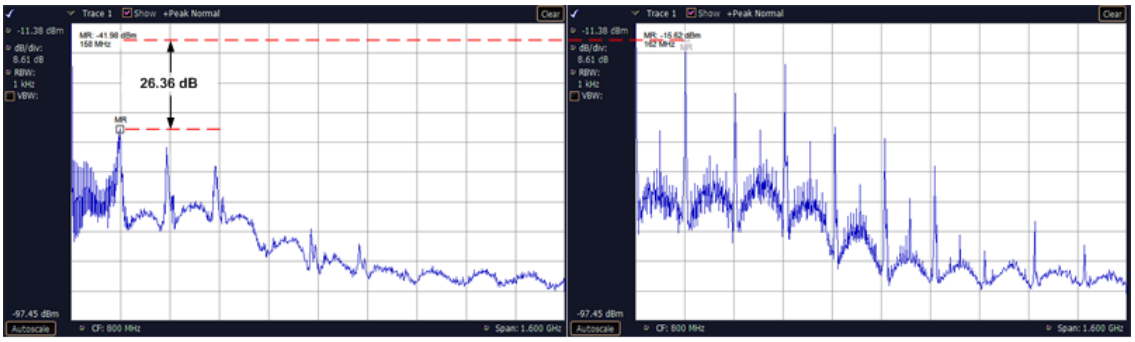
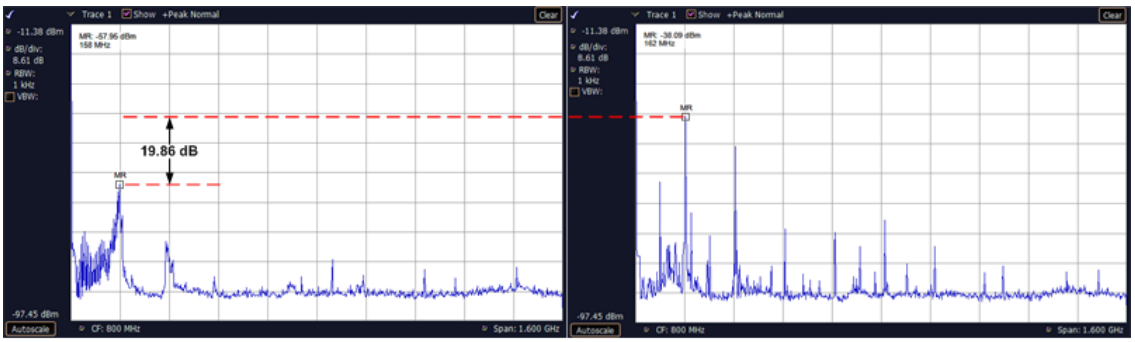


Figure 8.16 *Moonrake* chip adapter board



(a) GALS working mode (b) Synchronous working mode

Figure 8.17 Spectral peaks of core-VDD measured at connector VDD_AE22 (*RBW = 1 kHz*)



(a) GALS working mode (b) Synchronous working mode

Figure 8.18 Spectral peaks of core-VDD measured at connector VDD_BOARD (*RBW = 1 kHz*)

8.3 Summary

This chapter presents the implementation and measurements of the *Lighthouse* and the *Moonrake* chips in great detail. The industry-relevant design examples and the state-of-the-art fabricate processes highlight the practical significance of the work. After great effort, we have achieved the first-silicon-success of both chips with full functionality.

The on-die parallel integration of the GALS and the synchronous designs allows an objective comparison in performance. The experimental results have well manifested the feasibility and the advantages of our GALS design methodology. The measurements on both chips prove the benefits of power-balanced partitioning for spectral peak attenuation of switching noise. The GALS performance also matches our analytical study. Even more interestingly, the *Moonrake* chip demonstrates, for the first time on silicon, the area and power efficiency of GALS integration.

There is never a perfectly objective experiment, although we have strived to make the measurements reliable and fair. This is in particular the case for our investigation of switching noise. We discovered and quantified the potential impacts of probe location and distance on noise measurement, which were neglected unfortunately in the development. These experiences will be very valuable for our future work.

Hardware efficiency of a GALS design is achievable, only in the case that the asynchronous interface circuits are sufficiently small compared to the synchronous functional modules. Given the dual-clock FIFOs typically used for instance, the GALS infrastructure often accounts for a 5% to 10% cost of the entire design. Obviously, this is difficult to be compensated at the system level. The pausible clocking based GALS design derived in our work provides a lightweight alternative. It is even possible, as the *Moonrake* chip exhibited, to lower the hardware costs of a system. In general, we believe that the GALS design by pausible clocking can be applied with negligible overheads.

Chapter 9

Conclusions

This thesis presents an in-depth study on GALS design by pausible clocking. The challenges of asynchronous interconnect design and GALS system partitioning are both addressed systematically and analytically. The developed design methodology has been validated by the practical chip implementation and measurements. In the following two aspects, the scientific contributions of this work can be summarized.

First, a high-throughput circuit design for loosely-coupled GALS data link is proposed. Its performances on synchronization reliability, data throughput and hardware overhead are thoroughly analyzed. For sub-cycle clock tree delays, the ME (Mutually-Exclusive) Latching scheme proves to be metastability free. Continuous data transfer is achievable, under the throughput-tolerant conditions of clock-insertion and port-interconnect delays. Only marginal costs of power and silicon area are required.

Second, the power-consumption balanced GALS partitioning is introduced to cope with digital switching noise. A complete spectrum analysis of on-chip supply current is given. For M GALS blocks with plesiochronous clocking, a reduction of $20lgM\ dB$ is theoretically derived, and silicon measured as well, on the fundamental clock frequency. Efficiency and generality account for its significance in practical applications compared with the existing low-noise synchronous design techniques.

The developed GALS design methodology thus paves the way of exploring more advanced techniques of system integration. Today, a design evolution of on-chip interconnects is rapidly progressing. With the dramatic increase of working frequency and system complexity, the traditional bus-based topologies turn to be inefficient for SoCs. The Network-on-Chip (NoC) design promises an alternative of performance, scalability and robustness [92] [93]. Interconnection between nodes goes through a set of switches, and data transmission is carried out in packets. GALS design provides one of the backbone techniques of NoCs in the physical layer. It allows each node to operate at a clock frequency optimum for the functionality. Dual-clock FIFOs are often employed in NoC design for synchronization. Typically they give rise to significant hardware costs due to the large amount of GALS blocks accommodated in a NoC. As a lightweight scheme of synchronization, pausible clocking based GALS design deserves our attention in future applications. An example of the GALS NoCs by pausible clocking has been reported in [94]. The dynamic voltage and frequency scaling (DVFS) technique was also applied in their work for power saving [95]. Note that, our proposed data link design well supports the GALS NoCs with DVFS. More important, its performance and hardware efficiency will further advance the practical applications of NoC design.

Bibliography:

[1] E. G. Friedman, "Clock distribution networks in synchronous digital integrated circuits," in *Proc. of the IEEE*, vol. 89, no. 5, 2001, pp. 665-692.

[2] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, V. De, "Parameter variations and impact on circuit and microarchitecture," in *Proc. 40th Design Automation Conference* (DAC), 2003, pp. 338-342.

[3] R. Saleh, S. Wilton, S. Mirabbasi, A. Hu, M. Greenstreet, G. Lemieux, P. P. Pande, C. Grecu, A. Ivanov, "System-on-Chip: reuse and integration," in *Proc. of the IEEE*, vol. 94, no. 6, 2006, pp. 1050-1069.

[4] D. A. Huffman, "The synthesis of sequential switching circuits," in *J. Franklin Inst.*, 1954, pp. 161-190, 275-303.

[5] D. E. Muller and W. S. Bartky, "A theory of asynchronous circuits," in *Proc. Intl. Symp. on the Theory of Switching*, 1957, pp. 204-243.

[6] I. E. Sutherland, "Micropipelines," in *Communications of the ACM*, 1989, pp. 720-738.

[7] J. Sparsø, S. Furber, "Principles of asynchronous circuit design – a systems perspective," Kluwer Academic Publishers, 2001.

[8] A. Taubin, J. Cortadella, L. Lavagno, A. Kondratyev, Ad. Peeters, "Design automation of real-life asynchronous devices and systems," in *J. Foundations and Trends in Electronic Design Automation*, vol. 2, no. 1, 2007, pp. 1-133.

[9] D. M. Chapiro, "Globally-asynchronous locally-synchronous systems," PhD thesis, Stanford University, 1984.

[10] M. J. Stucki, J. J. Cox, "Synchronization strategies," in *Proc. 1st Caltech Conf. on Very Large Scale Integration*, 1979, pp. 375-393.

[11] R. Ginosar, "Metastability and synchronizers: a tutorial," in *IEEE Design & Test of Computers*, vol. 28, no. 5, 2011, pp. 23-35.

[12] P. Teehan, M. Greenstreet, and G. Lemieux, "A survey and taxonomy of GALS design styles," in *IEEE Design & Test of Computers*, vol. 24, no. 5, 2007, pp. 418-428.

[13] I. Miro-Panades, F. Clermidy, P. Vivet, and A. Greiner, "Physical implementation of the DSPIN Network-on-Chip in the FAUST architecture," in *Proc. 2nd ACM/IEEE Intl. Symp. on Networks-on-Chip* (NOCS), 2008, pp. 139-148.

[14] T. Chaney, and C. Molnar, "Anomalous behavior of synchronizer and arbiter circuits," in *IEEE Trans. Computers*, vol. C-22, no. 4, 1973, pp. 421-422.

[15] C. Dike, and E. Burton, "Miller and noise effects in a synchronizing flip-flop," in *IEEE J. Solid-State Circuits*, vol. 34, no. 6, 1999, pp. 849-855.

[16] S. Beer, R. Ginosar, M. Priel, R. Dobkin, and A. Kolodny, "The devolution of synchronizers," in *Proc. 16th IEEE Intl. Symp. on Asynchronous Circuits and Systems* (ASYNC), 2010, pp. 94-103.

[17] J. N. Seizovic, "Pipeline synchronization," in *Proc. 1st IEEE Intl. Symp. on Asynchronous Circuits and Systems* (ASYNC), 1994, pp. 87-96.

[18] R. Dobkin, and R. Ginosar, "Two-phase synchronization with sub-cycle latency," in *Integration, the VLSI journal*, vol. 42, no. 3, 2009, pp. 367-375.

[19] C. Dike, "Synchronization tutorial," presented in 6th *IEEE Intl. Symp. on Asynchronous Circuits and Systems* (ASYNC), 2000.

[20] R. Ginosar, "Fourteen ways to fool your synchronizer," in *Proc. 9th IEEE Asynchronous Circuits and Systems* (ASYNC), 2003, pp. 89-96.

[21] C. Cumming, "Simulation and synthesis techniques for asynchronous FIFO design," in *Synopsys Users Group Conference* (SNUG), San Jose, CA, 2002.

[22] C. Cumming, "Clock domain crossing (CDC) design & verification techniques using SystemVerilog," in *Synopsys Users Group Conference* (SNUG), Boston, 2008.

[23] A. Oberai, K A. Prasad, S. Kurupati, "Method and apparatus for generating Gray code for any even count value to enable efficient pointer exchange mechanisms in asynchronous FIFO's," U.S. Patent Application 10/926, 2004.

[24] I. Mrio Panades, A. Greiner, "Bi-synchronous FIFO for synchronous circuit communication well suited for Network-on-Chip in GALS architectures," in *Proc. 1st ACM/IEEE Intl. Symp. on Networks-on-Chip* (NOCS), 2007, pp. 83-94.

[25] Y. Thonnart, E. Beigne, P. Vivet, "Design and implementation of a GALS adapter for ANoC based architectures," in *Proc. 15th IEEE Intl. Symp. on Asynchronous Circuits and Systems* (ASYNC), 2009, pp. 13-22.

[26] T. Chelcea, S. Nowick, "Robust interface for mixed-timing systems," in *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 8, 2004, pp. 857-873.

[27] D. M. Chapiro, "Reliable high-speed arbitration and synchronization," in *IEEE Trans. on Computers*, vol. C-36, no. 10, 1987, pp. 1251-1255.

[28] K. Yun, R. Donohue, "Pausible clocking: a first step toward heterogeneous system," in *Proc. 14th IEEE Intl. Conf. on Computer Design* (ICCD), 1996, pp. 118-123.

[29] R. Mullins, S. Moore, "Demystifying data-driven and pausible clocking scheme," in *Proc. 13th IEEE Intl. Symp. on Asynchronous Circuits and Systems* (ASYNC), 2007, pp. 175-185.

[30] J. Kessels, A. Peeters, P. Wielage, SJ. Kim, "Clock synchronization through handshake signaling," in *Proc. 8th IEEE Symp. on Asynchronous Circuits and Systems* (ASYNC), 2003, pp. 59-68.

[31] S. Moore, G. Taylor, R. Mullins, P. Robinson, "Point to point GALS interconnect," in *Proc. 8th IEEE Intl. Symp. on Asynchronous Circuits and Systems* (ASYNC), 2002, pp. 69-75.

[32] D. Bormann, P. YK. Cheung, "Asynchronous wrapper for heterogeneous systems," in *Proc. 15th IEEE Intl. Conf. on Computer Design* (ICCD), 1997, pp. 307-314.

[33] S W. Moore, G. Taylor, P A. Cunningham, R. Mullins, P. Robinson, "Using stoppable clocks to safely interface asynchronous and synchronous subsystems," in *Proc. Asynchronous Interface Workshop* (AINT), 2000, pp. 20-23.

[34] S W. Moore, G. Taylor, P A. Cunningham, R. Mullins, P. Robinson, "Self calibrating clocks for globally asynchronous locally synchronous systems," in *Proc. 18th IEEE Intl. Conf. on Computer Design* (ICCD), 2000, pp. 73-78.

[35] E. Beigne, F. Clermidy, S. Miermont, P. Vivet, "Dynamic voltage and frequency scaling architecture for units integration within a GALS NoC," in *Proc. 2nd ACM/IEEE Intl. Symp. on Networks-on-Chip* (NOCS), 2008, pp. 129-138.

[36] S. Oetiker, T. Villiger, F K. Gurkaynak, H. Kaeslin, N. Felber, W. Fichtner, "High resolution clock generators for globally-asynchronous locally-synchronous designs," in *Proc. 2nd ACiD-WG Workshop of the European Commission's FP-5*, 2002.

[37] S. Temple, S B. Furber, "On-chip timing reference for self-timed microprocessor," in *IEEE Electronics Letters*, vol. 36, no. 11, 2000, pp. 942-943.

[38] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Yakovlev, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers," in *IEICE Transactions on information and Systems*, 1997, pp. 315-325.

[39] J. Muttersbach, T. Villiger, W. Fichtner, "Practical design of globally-asynchronous locally-synchronous systems," in *Proc. 6th IEEE Intl. Symp. on Asynchronous Circuits and Systems* (ASYNC), 2000, pp. 52-59.

[40] R. Dobkin, R. Ginosar, C. P. Sotiriou, "Data synchronization issues in GALS SoCs," in *Proc. 10th IEEE Symp. on Asynchronous Circuits and Systems* (ASYNC), 2004, pp. 170-179.

[41] C. L. Seitz, "System Timing," in *Introduction to VLSI Systems, edited by C. Mead and L. Conway,* Addison-Wesley, 1980, pp. 218-262.

[42] A. J. Martin, "Programming in VLSI: From communicating processes to delay-insensitive circuits," in *Developments in Concurrency and Communication*, *edited by C.A.R. Hoare,* Addison-Wesley, 1990, pp. 1-64.

[43] A. C. Davies, "Metastability in latches, arbiters and data-convertors," in *Proc. Symp. on Signals, Circuits and Systems* (SCS), 1999, pp. 6-7.

[44] G. Fuchs, M. Fugger, A. Steininger, "On the threat of metastability in an asynchronous fault-tolerant clock generator scheme," in *Proc. 15th IEEE Symp. on Asynchronous Circuits and Systems*, 2009, pp. 127-136.

[45] A E. Sjogren, C J. Myers, "Interfacing synchronous and asynchronous modules within a high-speed pipeline," in *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 15, 2000, pp. 573-583.

[46] R. Dokbin, R. Ginosar, C P. Sotiriou, "High rate data synchronization in GALS SoCs," in *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 10, 2006, pp. 1063-1074.

[47] J. Mekie, S. Chakraborty, D. K. Sharma, "Evaluation of pausible clocking for interfacing high speed IP cores in GALS framework," in *Proc. 17th IEEE Intl. Conf. on VLSI Design*, 2004, pp. 559-564.

[48] S. Dasgupta, A. Yakovlev, "Modeling and performance analysis of GALS architectures," in *Proc. IEEE Intl. Sympo. on System-on-Chip* (SOC), 2006, pp. 1-4.

[49] R. Dobkin, R. Ginosar, "Two-phase synchronization with sub-cycle latency," in *Integration, the VLSI Journal*, vol. 42, no. 3, 2009, pp. 367-375.

[50] K. Niyogi, D. Marculescu, "System level power and performance modeling of GALS point-to-point communication interfaces," in *Proc. Intl. Symp. on Low Power Electronics and Design (ISLPED)*, 2005, pp. 381-386.

[51] Z. Zhang, J. Garside, "Performance analysis of two synchronizers," presented in *20th UK Asynchronous Forum*, 2008

[52] M. Singh, S. M. Nowick, "MOUSETRAP: High-speed transition-signaling asynchronous pipeline," in *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 6, 2007, pp. 684-698.

[53] S. M. Nowick, M. Singh, "High-performance asynchronous pipelines: an overview," in *IEEE Design & Test of Computer*, vol. 28, no. 5, 2011, pp. 8-22.

[54] T. Villiger, H. Käslin, F. Gürkaynak, S. Oetiker, W. Fichtner, "Self-timed Ring for Globally-Asynchronous Loccally-Synchronous Systems," in *Proc. 9th IEEE Intl. Symp. on Asynchronous Circuit and Systems* (ASYNC), 2003, pp. 1-10.

[55] F. Gurkaynak, S. Oetiker, H. Kaeslin, N. Felber, W. Fichtner, "Improving DPA security by using globally-asynchronous locally-synchrnous systems," in *Proc. 31th European Solid-State Circuits Conference* (ESSCIRC), 2005, pp. 407-410.

[56] M. Swaminathan, A. E. Engin, "Power integrity modeling and design for semiconductors and systems," Prentice Hall, 2007.

[57] X. Aragones, J. L. Gonzalez, A. Rubio, "Analysis and solutions for switching noise coupling in mixed-signal ICs," Kluwer Academic Pub, 1999.

[58] C. R. Paul, "Introduction to electromagnetic compatibility," 2nd Edition, John Wiley & Sons, 2006.

[59] R. C. Frye, "Integration and electrical isolation in CMOS mixed-signal wireless chips," in *Proc. of the IEEE*, vol. 89, no. 4, 2001, pp. 444-455.

[60] M. Badaroglu, P. Wambacq, G. Van der Plas, S. Donnay, G. G. E. Gielen, H. J. De Man, "Digital ground bounce reduction by supply current shaping and clock frequency modulation," in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 1, 2005, pp. 65-76.

[61] M. Badaroglu, K. Tiri, G. Van der Plas, P. Wambacq, I. Verbauwhede, S. Donnay, G. G. Gielen, H. J. De Man, "Clock-skew-optimization methodology for substrate-noise reduction with supply-current folding," in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 6, 2006, pp. 1146-1154.

[62] M. Mendez, D. Mateo, A. Rubio, J. L. Gonzalez, "Analytical and experimental verification of substrate noise spectrum for mixed-signal ICs," in *IEEE Trans. on Circuits and Systems I: Regular Paper*, vol.53, no. 8, 2006, pp. 1803-1815.

[63] G. Boselli, G. Trucco, V. Liberali, "Properties of digital switching currents in fully CMOS combinational logic," in *IEEE Trans. on Very Large Scale Integration (VLSI) systems*, vol. 18, no. 12, 2010, pp. 1625-1638.

[64] R. Jakushokas, M. Popovich, A.V. Mezhiba, S. Kose, E. G. Friedman, "Power distribution networks with on-chip decoupling capacitors," 2nd Edition, Springer, 2011.

[65] P. Vuillod, L. Benini, A. Gogliolo, G. De Micheli, "Clock-skew optimization for peak current reduction," in *Proc. Intl. Symp. on Low Power Electronics and Design (ISLPED)*, 1996, pp. 355-360.

[66] R. Hyman, N. Ranganathan, T. bingel, D. T. Vo, "A clock control strategy for peak power and RMS current reduction using path clustering," in *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 2, 2013, pp. 259-269.

[67] K. B. Hardin, J. T. Fessler, D. R. Bush, "Spread spectrum clock generation for the reduction of radiated emissions," in *Proc. IEEE Intl. Symp. on Electromagnetic Compatibility* (EMC), 1994, pp. 227-231.

[68] Y. Matsumoto, K. Fujii, A. Sugiura, "An analytical method for determining the optimal modulating waveform for dithered clock generation," in *IEEE Trans. on Electromagnetic Compatibility*, vol. 47, no. 3, 2005, pp. 577-584.

[69] J. Kim, D. G. Kam, P. J. Jun, J. Kim, "Spread spectrum clock generator with delay cell array to reduce electromagnetic interference," in *IEEE Trans. on Electromagnetic Compatibility*, vol. 47, no. 4, 2005, pp. 908-920.

[70] S. Damphousse, K. Ouici, A. Rizki, M. Mallinson, "All digital spread spectrum clock generator for EMI reduction," in *IEEE Intl. Solid-State Circuits Conference* (ISSCC), 2006, pp. 962-971.

[71] F. Pareschi, G. Setti, R. Rovatti, "A 3-GHz serial ATA spread-spectrum clock generator employing a chaotic PAM modulation," in *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 57, no. 10, 2010, pp. 2577-2587.

[72] J. R. Carson, "Notes on the theory of Modulation," in *Proc. Institute of Radio Engineers*, vol. 10, no. 1, 1922, pp. 57-64.

[73] E. Grass, F. Winkler, M. Krstic, A. Juilius, C. Stahl, M. Piz, "Enhanced GALS techniques for datapath applications," in *Proc. 15th Intl. Conf. on Integrated Circuit and System Design: Power and Timing Modeling, Optimization and Simulation* (PATMOS), 2005, pp. 581-590.

[74] A. Hemani, T. Meincke, S. Kumar, A. Postula, T. Olsson, P. Nilsson, J. Oberg, P. Ellervee, D. Lundqvist, "Lower power consumption in clock by using globally asynchronous locally synchronous design style," in *Proc. 36th Design Automation Conference* (DAC), 1999, pp. 873-878.

[75] R. M. Fuhrer, S. M. Nowick, "Sequential optimization of asynchronous and synchronous finite-state machines: algorithms and tools," Kluwer Academic Publishers, 2001.

[76] F. K. Gurkaynak, S. Oetiker, T. Villiger, N. Felber, H. Kaeslin, W. Fichtner, "On the GALS design methodology of ETH Zurich," presented in *1st Workshop on Formal Verification of GALS Design* (FMGALS), 2003.

[77] L. Janin, D. Edwards, "AsipIDE: a graphical framework to design and debug GALS system through simulation and prototyping," available at http://www.minatec.org/nocs2010.

[78] C. Wolf, S. Zeidler, M. Krstic, R. Kraemer, "Overview on ATE test and debugging method for asynchronous circuits," in *Proc. 12th Intl. Workshop on Microprocessor Test and Verification* (MTV), 2011, pp. 16-21.

[79] H. Hulgaard, S. M. Burns, G. Borriello, "Testing asynchronous circuits: a survey," in *Integration, the VLSI Journal*, vol. 19, no. 3, 1995, pp. 111-131.

[80] F. Gurkaynak, T. Villiger, S. Oetiker, N. Felber, H. Kaeslin, W. Fichtner, "A functional test methodology for globally-asynchronous locally-synchronous systems, " in *Proc. 8th IEEE Intl. Symp. on Asynchronous Circuits and Systems* (ASYNC), 2002, pp. 181-189.

[81] M. W. Heath, W. P. Burleson, I. G. Harris, "Synchro-tokens: eliminating nondeterminism to enable chip-level test of globally-asynchronous locally-synchronous SoCs," in *Proc. Design, Automation and Test in Europe Conference and Exhibition* (DATE), 2004, pp. 410-415.

[82] F. K. Gurkaynak, S. Oetiker, H. Kaeslin, N. Felber, W. Fichtner, "GALS at ETH Zurich: success or failure?" in *Proc. 12th IEEE Intl. Symp. on Asynchronous Circuits and Systems* (ASYNC), 2006, pp. 1-10.

[83] L. P. Carloni, K. L. McMillan, A. Saldanha, A. L. Sangiovanni-Vincentelli, "A methodology for correct-by-construction latency insensitive design," in *IEEE/ACM Intl. Conf. on Computer-Aided Design* (ICCAD), 1999, pp. 309-315.

[84] J. Carmona, J. Cortadella, M. Kishinevsky, A. Taubin, "Elastic Circuits," in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol.28, no. 10, 2009, pp. 1437-1455.

[85] A. G. Stove, "Linear FMCW radar techniques," in *IEE Proc. for Radar and Signal Processing*, vol. 139, no. 5, 1992, pp. 343-350.

[86] Y. M. Sun, M. Marinkovic, G. Fischer, W. Winkler, W. Debski, S. Beer, T. Zwick, M. G. Girma, J. Hasch, C. J. Scheytt, "A low-cost miniature 120GHz SiP FMCW/CW radars sensor with software linearization," in *IEEE Intl. Solid-State Circuits Conference* (ISSCC), 2013, pp. 148-149.

[87] A. M. Despain, "Fourier Transform computations using CORDIC iterations," in *IEEE Trans. on Computers*, vol. 23, 1973, pp. 993-1001.

[88] R. van Nee, R. Prasad, "OFDM for wireless multimedia communications," Artech House Publishers, 1999.

[89] E. Grass, F. Herzel, M. Piz, K. Schmalz, Y. Sun, S. Glisic, M. Krstic, K. Tittelbach, M. Ehrig, W. Winkler, C. Scheytt, R. Kraemer, "60 GHz SiGe-BiCMOS radio for OFDM transmission," in *Proc. IEEE Intl. Symp. on Circuits and Systems* (ISCAS), 2007, pp. 1979-1982.

[90] M. Krstic, M. Piz, E. Grass, "60 GHz datapath processor for 1 Gbit/s," in *Proc. 16th IFIP/IEEE Intl. Conf. on Very Large Scale Integration* (VLSI-SOC), 2008, pp. 156-159.

[91] M. Piz, E. Grass, "A synchronization scheme for OFDM-based 60 GHz WPANs," in *Proc. 18th Intl. Symp. on Personal, Indoor and Mobile Radio Communications* (PIMRC), 2007, pp. 1-5.

[92] W. J. Dally, B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proc. 38th Design Automation Conference* (DAC), 2001, pp. 684-689.

[93] L. Benini, G. De Micheli, "Networks on chips: a new SoC paradigm," in *IEEE Computer*, vol.35, no. 1, 2002, pp. 70-78.

[94] D. Lattard, E. Beigne, C. Bernard, C. Bour, F. Clermidy, Y. Durand, J. Durupt, D. Varreau, P. Vivet, P. Penard, A. Bouttier, F. Berens, "A telecom baseband circuit based on an asynchronous Network-on-Chip," in *IEEE Intl. Solid-State Circuits Conference* (ISSCC), 2007, pp. 258-601.

[95] E. Taples, D. Marculescu, "Toward a multiple clock/voltage island design style for power-aware processors," in *IEEE Trans. on Very Large Scale Integration (VLSI) systems*, vol. 13, no. 5, 2005, pp. 591-603.

Appendix:

A. MATLAB code of current spectrum analysis for GALS design

```
clear;
clc;

%set default sampling period of 10 ps
sample_period=11*10^(-12);
%derive default sampling frequency in Hz
sample_frequency=1/sample_period;

%set clock parameters
%baseline clock frequency in Hz
clock_frequency_baseline=input('baseline clock frequency (MHz) =
')*10^6;
%baseline clock period in s
clock_period_baseline=1/clock_frequency_baseline;
%baseline clock period in sample number
clock_period_baseline=round(clock_period_baseline/sample_period);
%baseline clock frequency in Hz
clock_frequency_baseline=sample_frequency/clock_period_baseline;

%set parameters for reference triangular current pattern
%corner time in percentage
pattern_corner_time=input('corner time (%) of reference current
pattern = ')/100;
%corner time in sample number
pattern_corner_time=floor(pattern_corner_time*clock_period_baseline);
%corner peak in mA
pattern_corner_peak=input('corner peak (mA) of reference current
pattern = ')/1000;
%corner point number
pattern_corner_numb=length(pattern_corner_time(1,:));

%set current pattern number
pattern_numb=input('number of current pattern = ');
%set maximum ratio on delay spread of current pattern
pattern_spread_delay_max=input('max ratio of delay spread on current
pattern = ');
%set maximum ratio on power spread of current pattern
pattern_spread_power_max=input('max ratio of power spread on current
pattern = ');
%set step in delay spread
pattern_spread_delay_step=input('step of delay spread on current
pattern = ');
%set step in power spread
pattern_spread_power_step=input('step of power spread on current
pattern = ');

%clock number
clock_numb=pattern_numb;
%clock frequency drifting in percentage
clock_frequency_drifting=input('clock frequency drifting (%) matrix =
')/100;
```

```
%clock frequency in Hz
clock_frequency=clock_frequency_baseline*(1+clock_frequency_drifting);
%clock period in s
clock_period=1./clock_frequency;
%clock period
clock_period=round(clock_period/sample_period);
%clock frequency in Hz
clock_frequency=sample_frequency./clock_period;

%set the number of harmonics for comparing the spectral peaks
clock_harmonics_numb=input('the number of clock harmonics to be
compared in spectrum = ');
%set power threhold for determining dominant ac harmonics
waveform_power_dominant=input('set dominant ac power (%) on current
waveform = ')/100;

iteration=1;
row=1;
for i=1:(pattern_spread_delay_max-1)/(pattern_spread_delay_step-
1):pattern_spread_delay_max %sweep in time axis
    col=1;
    for j=1:(pattern_spread_power_max-1)/(pattern_spread_power_step-
1):pattern_spread_power_max %sweep in power axis

        %set current pattern
        %pattern length
        pattern_length=clock_period_baseline*1000;
        %initialization
        current_pattern=zeros(pattern_numb,pattern_length);
        %initialization
pattern_corner_time_tmp=zeros(pattern_numb,pattern_corner_numb);
pattern_corner_peak_tmp=zeros(pattern_numb,pattern_corner_numb);

        %step 1: initial charge averaging over clock domains
        if row==1&&col==1
            pattern_corner_peak=pattern_corner_peak/pattern_numb;
        end

        %step 2: spread in power according to j
        for k=1:pattern_numb
            %tmp corner time
            pattern_corner_time_tmp(k,:)=pattern_corner_time;
            %tmp corner peak
pattern_corner_peak_tmp(k,:)=pattern_corner_peak*pattern_numb/(sum((1:
(j-1)/(pattern_numb-1):j))+(j==1)*pattern_numb)*(1+(k-1)*(j-
1)/(pattern_numb-1));
        end

        %step 3: spread in delay according to i
        for k=1:pattern_numb
            %tmp corner time
pattern_corner_time_tmp(k,:)=floor(pattern_corner_time_tmp(k,:)*(1+(k-
1)*(i-1)/(pattern_numb-1)));
            %tmp corner peak
pattern_corner_peak_tmp(k,:)=pattern_corner_peak_tmp(k,:)/(1+(k-1)*(i-
1)/(pattern_numb-1));
        end
```

```
        for l=1:pattern_numb
            for m=1:pattern_corner_numb
                if m==1 %the first piece of wave starting from 0
                    for n=1:pattern_corner_time_tmp(l,1)
                        current_pattern(l,n)=(n-
1)*pattern_corner_peak_tmp(l,1)/pattern_corner_time_tmp(l,1);
                    end
                else %the other linear wave
                    for n=pattern_corner_time_tmp(l,m-
1)+1:pattern_corner_time_tmp(l,m)
current_pattern(l,n)=pattern_corner_peak_tmp(l,m-1)+(n-
pattern_corner_time_tmp(l,m-1)-1)*(pattern_corner_peak_tmp(l,m)-
pattern_corner_peak_tmp(l,m-1))/(pattern_corner_time_tmp(l,m)-
pattern_corner_time_tmp(l,m-1));
                    end
                end
            end
        end

        %total charge of current pattern, constant value expected
pattern_charge(row,col)=sum(sum(current_pattern.')*sample_period);

        %profile length
        profile_length=pattern_length;

        %current profile
current_profile_sync=sum([current_pattern;0*(1:profile_length)]);
        current_profile_gals=current_pattern;

        %periodic waveform of current profile in sync/gals mode
        %waveform length
        waveform_length=profile_length;

current_waveform_sync=zeros(1,waveform_length); %initialization
        for k=1:floor(waveform_length/clock_period_baseline)
            current_waveform_sync((k-
1)*clock_period_baseline+1:k*clock_period_baseline)=current_profile_sy
nc(1:clock_period_baseline);
        end

current_waveform_gals=zeros(clock_numb,waveform_length); %initialize
        for k=1:clock_numb
            for l=1:floor(waveform_length/clock_period(k))
                current_waveform_gals(k,(l-
1)*clock_period(k)+1:l*clock_period(k))=current_profile_gals(k,1:clock
_period(k));
            end
        end

        %DFT
        %perform dft only on the integer clock cycles
waveform_dft_sync(1:clock_period_baseline*floor(waveform_length/clock_
period_baseline))=fft(current_waveform_sync(1:clock_period_baseline*fl
oor(waveform_length/clock_period_baseline)));

waveform_dft_gals=zeros(waveform_length,clock_numb); %initialization
        for k=1:clock_numb
```

```
waveform_dft_gals(1:clock_period(k)*floor(waveform_length/clock_period
(k)),k)=fft(current_waveform_gals(k,1:clock_period(k)*floor(waveform_l
ength/clock_period(k))).');
        end

        %FT
        waveform_ft_sync=waveform_dft_sync*sample_period; %Ts-scaling
        waveform_ft_gals=waveform_dft_gals*sample_period;

        %magnitude spectrum of FT
        waveform_ft_gals=abs(waveform_ft_gals);
        waveform_ft_sync=abs(waveform_ft_sync);

        %magnitude envolop of FS spectrum
waveform_ft_sync=waveform_ft_sync*clock_frequency_baseline/floor(wavef
orm_length/clock_period_baseline); %scaling by fc/(number of clock
cycles)
        for k=1:clock_numb
waveform_ft_gals(:,k)=waveform_ft_gals(:,k)*clock_frequency(k)/floor(w
aveform_length/clock_period(k));
        end

        %magnitude envolop of FS spectrum in dBA
        waveform_ft_sync=20*log10(waveform_ft_sync);
        waveform_ft_gals=20*log10(waveform_ft_gals);

        %derive clock harmonics in sample number
clock_harmonics_sync=(1:clock_harmonics_numb)*floor(profile_length/clo
ck_period_baseline)+1; %k=n*N*fc/fs
clock_harmonics_gals=zeros(clock_numb,clock_harmonics_numb); %initial
for k=1:clock_numb
clock_harmonics_gals(k,:)=(1:clock_harmonics_numb)*floor(profile_lengt
h/clock_period(k))+1;
        end

        %spectral peaks at clock harmonics
        harmonics_peak_sync=waveform_ft_sync(clock_harmonics_sync);
harmonics_peak_gals=zeros(clock_numb,clock_harmonics_numb); %initial
harmonics_peak_gals(k,:)=waveform_ft_gals(clock_harmonics_gals(k,:),k);
        end

        %attenuation
        harmonics_peak_atte=harmonics_peak_sync-
max([harmonics_peak_gals;-inf*(1:clock_harmonics_numb)]);

        %power of current waveform in frequency domain
        %power at clock harmonics
        waveform_power_f_sync_ac=10.^(harmonics_peak_sync/20).^2*2;
        waveform_power_f_gals_ac=10.^(harmonics_peak_gals/20).^2*2;

        %power at DC component
        waveform_power_f_sync_dc=10^(waveform_ft_sync(1)/20)^2;
        waveform_power_f_gals_dc=10.^((waveform_ft_gals(1,:)/20)).^2;

        %total power
waveform_power_f_sync=sum(waveform_power_f_sync_ac)+waveform_power_f_s
ync_dc;
```

```
aveform_power_f_gals=sum(waveform_power_f_gals_ac.')+waveform_power_f_
gals_dc;

        %dominant ac harmonic
        for k=1:clock_harmonics_numb
            if
sum(waveform_power_f_sync_ac(1:k))<sum(waveform_power_f_sync_ac)*wavef
orm_power_dominant
                harmonics_peak_dominant=k+1;
            end
        end

        %ac power distribution over clock harmonics
waveform_power_distribution=waveform_power_f_sync_ac/sum(waveform_powe
r_f_sync_ac);

        %attenuation at clock fundamental frequency
        harmonics_peak_atte_f1(row,col)=harmonics_peak_atte(1);

        %maximum attenuation according to clock number
        harmonics_peak_atte_max(row,col)=20*log10(clock_numb);

        %arithmetic averaged attenuation on the dominant harmonics
harmonics_peak_atte_avg(row,col,1)=mean(harmonics_peak_atte(1:harmonic
s_peak_dominant));

        %variation of arithmetic averaged attenuation
        harmonics_peak_atte_var(row,col)=(sum((harmonics_peak_atte-
harmonics_peak_atte_avg(row,col,1)).^2)/harmonics_peak_dominant)^0.5;

        %weighted averaged attenuation on the total ac components
harmonics_peak_atte_avg(row,col,2)=sum(harmonics_peak_atte.*waveform_p
ower_distribution);

        %average attenuation according to power reduction
harmonics_peak_atte_avg(row,col,3)=10*log10(waveform_power_f_sync/max(
waveform_power_f_gals));

      hold off;

        iteration=iteration+1;

        col=col+1;

    end
    row=row+1;
end

surf(1:pattern_spread_power_step,1:pattern_spread_delay_step,harmonics
_peak_atte_f1);

xlim([1 pattern_spread_power_step]);
ylim([1 pattern_spread_delay_step]);

xlabel('max/min power ratio');
ylabel('max/min delay ratio');
```

B. Synthesis timing constraints of *Lighthouse* GALS FFT coprocessor

B.1 Input port controller

```
#-------------------------------------------------------------
# set driving/load constraints
#-------------------------------------------------------------

set_driving_cell -library ih13ugfsdsc_ss.db:ih13ugfsdsc_ss -lib_cell SDN_ND2B_2 -pin X –
input_transition_rise 0.2 -from_pin A -no_design_rule [all_inputs]

set_pin_cap [load_of [get_lib_pins ih13ugfsdsc_ss/SDN_ND2B_2/A]]

set_load [expr 32 * $pin_cap] [all_outputs]

set_max_fanout 4 PINTRANS2PHASE

set_max_transition 0.1 PINTRANS2PHASE

#-------------------------------------------------------------
# set maximum delay
#-------------------------------------------------------------

set_max_delay 0.30 -from [get_ports RP] -to [get_ports RI]

set_max_delay 0.30 -from [get_ports AI] -to [get_ports AP]

set_max_delay 0.20 -from [get_ports AI] -to [get_ports TA]
```

B.2 Output port controller

```
#-------------------------------------------------------------
# set driving/load constraints
#-------------------------------------------------------------

set_driving_cell -library ih13ugfsdsc_ss.db:ih13ugfsdsc_ss -lib_cell SDN_ND2B_2 -pin X -
input_transition_rise 0.2 -from_pin A -no_design_rule [all_inputs]

set_pin_cap [load_of [get_lib_pins ih13ugfsdsc_ss/SDN_ND2B_2/A]]

set_load [expr 32 * $pin_cap] [all_outputs]

set_max_fanout 4 DOUTTRANS2PHASE

set_max_transition 0.1 DOUTTRANS2PHASE

#-------------------------------------------------------------
# set maximum delay
#-------------------------------------------------------------

#set_max_delay 0.30 -from [get_ports RP] -to [get_ports RI]

#set_max_delay 0.30 -from [get_ports AI] -to [get_ports AP]

#set_max_delay 0.25 -from [get_ports AI] -to [get_ports TA]
```

B.3 Local clock generator

```
#-------------------------------------------------------------
# set RST/CLK constraints
#-------------------------------------------------------------

#set maximum delay from external reset signal to meet the removal/recovery timing constraint
set_max_delay -from [get_ports nRSTI] 2.5
set_false_path -from [get_ports nRSTI] -through [get_cells "u_D4/SLC_0/* u_D5_?/SLC_0/*
u_D6_?/SLC_0/* u_D8/SLC_0/*"]

#set internal clock signal
create_clock -period 16.0 -waveform {00.0 08.0} [get_pins u_MullerC/Y] -name LCLK

set_clock_latency -max 0.3 [all_clocks]
set_clock_latency -min 0.0 [all_clocks]
set_clock_uncertainty -setup 0.5 [all_clocks]
set_clock_uncertainty -hold  0.5 [all_clocks]
set_clock_transition -max 0.15 [all_clocks]
set_clock_transition -min 0.05 [all_clocks]

set_false_path -from [get_clocks nRst_CLK] -to [get_clocks LCLK]

set_false_path -from [get_clocks nRst_CLK] -through [get_cells "u_D4/SLC_0/*"]

set_false_path -from [get_clocks LCLK] -to [get_pins PDD_Sync_reg_0_/RD]

set_false_path -from [get_clocks LCLK] -through [get_pins PDD_Sync_reg_1_/Q]

set_false_path -to [get_pins u_MullerC/A]

set_false_path -to [get_pins u_MullerC/B]

set_false_path -from [get_ports nRSTI]

#-------------------------------------------------------------
# set driving/load constraints
#-------------------------------------------------------------

set_driving_cell -library ih13ugfsdsc_ss.db:ih13ugfsdsc_ss -lib_cell SDN_ND2B_2 -pin X -
input_transition_rise 0.2 -from_pin A -no_design_rule [all_inputs]

set_pin_cap [load_of [get_lib_pins ih13ugfsdsc_ss/SDN_ND2B_2/A]]

set_load [expr 32 * $pin_cap] [all_outputs]

set_max_fanout 8 $TOP_DESIGN

set_max_transition 0.25 $TOP_DESIGN

#-------------------------------------------------------------
# set force paths
#-------------------------------------------------------------

#supports static mode configuration only
set_false_path -from [get_ports CC[*]]
set_false_path -from [get_ports CLKID[*]]
```

B.4 Top level of GALS FFT core

```
#-----------------------------------------------------------------
# set attributes on reset networks
#-----------------------------------------------------------------

#set false path on input reset
set_false_path -from [get_ports rstn]

#set ideal network on input reset
set_ideal_network [get_ports rstn]
set_disable_timing [get_ports rstn]
set_dont_touch_network [get_ports rstn]

#set ideal networks on the synchronized GALS reset
set_ideal_network [get_pins u_lclkgen_gals?/nRst_Sync_reg_1_/Q]
set_disable_timing [get_pins u_lclkgen_gals?/nRst_Sync_reg_1_/Q]
set_dont_touch_network [get_pins u_lclkgen_gals?/nRst_Sync_reg_1_/Q]

#-----------------------------------------------------------------
# clock declaration
#-----------------------------------------------------------------

#create GALS local clocks
for {set BLK_NUM 0} {$BLK_NUM < 5} {incr BLK_NUM} {create_clock -period 16 -waveform {0 8}
[get_pins u_lclkgen_gals$BLK_NUM/CLKO] -name GALS_BLK$BLK_NUM\_CLK}

#create GALS ack clocks
#only the ack associated with input ports are declared as clock signals
#GALS block 1
for {set PORT_NUM 5} {$PORT_NUM < 10} {incr PORT_NUM} {create_clock -period 1.0 -
waveform {0 0.5} [get_pins u_lclkgen_gals0/ACK[$PORT_NUM]] -name
GALS_BLK0_ACK[$PORT_NUM]}

#GALS block 2
for {set PORT_NUM 2} {$PORT_NUM < 4}  {incr PORT_NUM} {create_clock -period 1.0 -waveform
{0 0.5} [get_pins u_lclkgen_gals1/ACK[$PORT_NUM]] -name GALS_BLK1_ACK[$PORT_NUM]}

#GALS block 3
for {set PORT_NUM 2} {$PORT_NUM < 4}  {incr PORT_NUM} {create_clock -period 1.0 -waveform
{0 0.5} [get_pins u_lclkgen_gals2/ACK[$PORT_NUM]] -name GALS_BLK2_ACK[$PORT_NUM]}

#GALS block 4
for {set PORT_NUM 2} {$PORT_NUM < 4}  {incr PORT_NUM} {create_clock -period 1.0 -waveform
{0 0.5} [get_pins u_lclkgen_gals3/ACK[$PORT_NUM]] -name GALS_BLK3_ACK[$PORT_NUM]}

#GALS block 5
for {set PORT_NUM 2} {$PORT_NUM < 4}  {incr PORT_NUM} {create_clock -period 1.0 -waveform
{0 0.5} [get_pins u_lclkgen_gals4/ACK[$PORT_NUM]] -name GALS_BLK4_ACK[$PORT_NUM]}

#create GALS gnt clocks
#only the gnt associated with input ports are declared as clock signals

#GALS block 1
for {set PORT_NUM 5} {$PORT_NUM < 10} {incr PORT_NUM} {create_clock -period 1.0 -
waveform {0 0.5} [get_pins u_lclkgen_gals0/GNT[$PORT_NUM]] -name
GALS_BLK0_GNT[$PORT_NUM]}
```

```
#GALS block 2
for {set PORT_NUM 2} {$PORT_NUM < 4}  {incr PORT_NUM} {create_clock -period 1.0 -waveform
{0 0.5} [get_pins u_lclkgen_gals1/GNT[$PORT_NUM]] -name GALS_BLK1_GNT[$PORT_NUM]}

#GALS block 3
for {set PORT_NUM 2} {$PORT_NUM < 4}  {incr PORT_NUM} {create_clock -period 1.0 -waveform
{0 0.5} [get_pins u_lclkgen_gals2/GNT[$PORT_NUM]] -name GALS_BLK2_GNT[$PORT_NUM]}

#GALS block 4
for {set PORT_NUM 2} {$PORT_NUM < 4}  {incr PORT_NUM} {create_clock -period 1.0 -waveform
{0 0.5} [get_pins u_lclkgen_gals3/GNT[$PORT_NUM]] -name GALS_BLK3_GNT[$PORT_NUM]}

#GALS block 5
for {set PORT_NUM 2} {$PORT_NUM < 4}  {incr PORT_NUM} {create_clock -period 1.0 -waveform
{0 0.5} [get_pins u_lclkgen_gals4/GNT[$PORT_NUM]] -name GALS_BLK4_GNT[$PORT_NUM]}

#----------------------------------------------------------------
# set attributes on SYNC/GALS clock networks
#----------------------------------------------------------------

#set insertion delay on GALS clocks

#CLK [0.2, 0.5]
set_clock_latency -max 0.5 [get_clocks GALS_BLK?_CLK]
set_clock_latency -min 0.2 [get_clocks GALS_BLK?_CLK]

#GNT [0.1, 0.2]
set_clock_latency -max 0.2 [get_clocks GALS_BLK?_GNT[*]]
set_clock_latency -min 0.1 [get_clocks GALS_BLK?_GNT[*]]

#ACK [0.0, 0.1]
set_clock_latency -max 0.1 [get_clocks GALS_BLK?_ACK[*]]
set_clock_latency -min 0.0 [get_clocks GALS_BLK?_ACK[*]]

#set clock skew
set_clock_uncertainty -setup 0.3 [all_clocks]
set_clock_uncertainty -hold  0.2 [all_clocks]

#set clock transition time
set_clock_transition -max 0.16 [all_clocks]
set_clock_transition -min 0.06 [all_clocks]

set_dont_touch_network [all_clocks]

#set ideal networks
# SYNC_SYST_CLK
set_ideal_network [get_ports clk]

# GALS_BLK?_CLK
set_ideal_network [get_pins "u_lclkgen_gals?/u_MullerC/Y"]

# GALS_BLK?_ACK
set_ideal_network [get_pins "u_lclkgen_gals?/u_D5_?/SLC_0/U2/X"]

# GALS_BLK?_GNT
set_ideal_network [get_pins "u_lclkgen_gals?/u_D6_?/SLC_0/U2/X"]
```

```
#-------------------------------------------------------------
# set maximal delay for interconnect
#-------------------------------------------------------------

#set maximum delay on RI from port controllers to local clock generators
set_max_delay 0.1 -from [get_pins "u_ip?_gals?/RI"] -to [get_pins "u_lclkgen_gals?/REQ*"]
set_max_delay 0.1 -from [get_pins "u_op?_gals?/RI"] -to [get_pins "u_lclkgen_gals?/REQ*"]

#set maximum delay on AI from local clock generators to port controllers
set_max_delay 0.1 -from [get_pins "u_lclkgen_gals?/ACK*"] -to [get_pins "u_ip?_gals?/AI"]
set_max_delay 0.1 -from [get_pins "u_lclkgen_gals?/ACK*"] -to [get_pins "u_op?_gals?/AI"]

#set maximum delay on RP from output port controllers to input port controllers
set_max_delay 0.1 -from [get_pins "u_op?_gals?/RP"] -to [get_pins "u_ip?_gals?/RP"]

#set maximum delay on AP from input port controllers to output port controllers
set_max_delay 0.1 -from [get_pins "u_ip?_gals?/AP"] -to [get_pins "u_op?_gals?/AP"]

#set maximum delay on the interconnect from GALS_BLK?_CLK to GALS_BLK?_ACK clock domains
for boundled-data constraints
set_max_delay 0.4 -from [get_clocks GALS_BLK?_CLK] -to [get_clocks GALS_BLK?_ACK]

#set maximum delay on the interconnect from GALS_BLK?_GNT to GALS_BLK?_CLK clock domains
for setup time requirement
set_max_delay 0.1 -from [get_clocks GALS_BLK?_GNT] -to [get_clocks GALS_BLK?_CLK]

#-------------------------------------------------------------
# set false paths
#-------------------------------------------------------------

#false paths across clock domains

#from SYNC_SYST_CLK to GALS_BLK?_CLK/ACK/GNT
set_false_path -from [get_clocks SYNC_SYST_CLK] -to [get_clocks GALS_BLK?_*]

#from GALS_BLK?_CLK/ACK/GNT to SYNC_SYST_CLK
set_false_path -from [get_clocks GALS_BLK?_*] -to [get_clocks SYNC_SYST_CLK]

#from GALS_BLK?_CLK to GALS_BLK?_ACK
set_false_path -from [get_clocks GALS_BLK?_CLK] -to [get_clocks GALS_BLK?_ACK[*]]

#from GALS_BLK?_CLK to GALS_BLK?_GNT
set_false_path -from [get_clocks GALS_BLK?_CLK] -to [get_clocks GALS_BLK?_GNT[*]]

#from GALS_BLK?_ACK to GALS_BLK?_GNT
set_false_path -from [get_clocks GALS_BLK?_ACK[*]] -to [get_clocks GALS_BLK?_GNT[*]]

#from GALS_BLK?_GNT to GALS_BLK?_CLK
set_false_path -from [get_clocks GALS_BLK?_GNT[*]] -to [get_clocks GALS_BLK?_CLK]

#false paths from asynchronous handshaking signals

#from TA of input ports to GALS_BLK?_GNT
set_false_path -from [get_pins "u_ip?_gals?/TA"]
```